Author:
Rees-Garbutt, Joshua P

Title:
Minimal Genome Design and Engineering

Algorithms and whole-cell Models

Minimal Genome Design and Engineering: Algorithms and whole-cell Models

Joshua Rees-Garbutt

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of Doctor of Philosophy in the Faculty of Life Sciences.

School of Biological Sciences,
September 2020

55,540 words

Abstract

Tailoring entire genomes to produce custom-made cells is now on the horizon. This level of control and understanding has been a goal for biologists since the publication of the first genome sequence in 1977. The field of genetics has expanded and grown since, advancing with the progress of genetic sequencing, synthesis, and editing, and with the discipline of synthetic biology emerging following the millennium. The editing and sequencing barriers to designing cells reduced dramatically in the early 2010s, with the publication of CRISPR-cas9 and the development of the MinION sequencer.

However, entire genome design still alludes us. Genome engineering progresses by systematic comparison of experimental results. The complexity and lack of knowledge of gene interactions still causes unexpected results. Libraries of genetic knockouts can only scale to encompass all possible double gene knockouts, before becoming economically infeasible, restricting data collection.

The publication of the first whole-cell model in 2012 (for *Mycoplasma genitalium*), combined with the availability and advancement of supercomputers from the mid-2000s, offers a way to tackle the complexity of gene interactions. It also allows genome engineers the possibility to emulate the work of metabolic engineers, who have an established cycle of *in-silico* design and *in-vivo* editing for their more constrained, sub-cellular systems.

In this thesis, I design entire genomes *in-silico* by producing a genome design algorithm to guide thousands of whole-cell model simulations running on supercomputers. I also simulate theoretical minimal genomes that have never been tested; and produce preliminary data from the second whole-cell model (for *Escherichia coli*).

This thesis aims to show that it is now possible to design entire genomes *in-silico*. The combination of *in-silico* (genome design algorithms, whole-cell models, supercomputers) and *in-vivo* (CRISPR-cas9 genetic editing, MinION genetic sequencing) components means the stage is now set for the *in-vivo* production of tailored genomes.

To my Wife, whose love for life outside of work is contagious.

To my Mum, whose belief in me is boundless.

To David, Georgia, Oska, Maeve, for being raucous and keeping me grounded.

To my Dad, two promises left.

Acknowledgements

I would like to thank my supervisors, Claire Grierson and Lucia Marucci. You have taught, mentored, and supported me into becoming a published scientist; an aspiration of mine that became an ambition and then a reality under your leadership. You took a chance on me and have changed the course of my life. I will always be grateful.

I thank my collaborator, Oliver Chalkley. An unlikely pairing of a blue-sky thinker and a goal-driven monomaniacal became a frictionless collaboration, with mutual trust and work ethic, and a unified vision. I have missed working with you. You taught me a lot about how others think and how I should collaborate and assemble teams in the future.

I thank my other group members and co-authors, Oliver Purcell for your experience, support, and evisceration of my drafts (between you and Claire you have made me a semi-competent scientific writer), and Sophie Landon for her patient explanations, her skill at data visualisation, and willingness to point out parts of Biology that aren't up to standard.

I thank Andy Bailey and Marc Holderied who selflessly and kindly supported my PhD application and return to academia, without whom I would not be here. I also thank the EPSRC for providing my studentship.

I thank Beth, Brogan, Helen, Ioan, and Emily for keeping me sane and something approaching happy through the hours in the Life Sciences building. Thank you for the good company and the black humour during the late nights, and for trying (though failing) to ward me off Red Bull.

I would like to thank my Mum, my fiancée-turned-wife Jennifer*, the new basic social unit (brother David, sister-in-law Georgia, dog Oska, baby Maeve Bronwyn), the family (Michael, the Barretts, Garbutts, Groves's, McKeowns, Morris's, Phillips's, Rees's, Ribbons's), the friends (Alex, Anna, Henry, Lydia, the LGGS girls and boys crews, and their associated partners), and the guinea pigs (Bumblebee, Honey, Nutmeg). For all the love, support, interest in my work, and all the reminders that it does not matter and no one cares. Without you all, life would just be work and there would be no light.

*who made co-organising a from-scratch wedding during a PhD fun!

Author’s Declaration

I declare that the work in this dissertation was carried out in accordance with the requirements of the University's Regulations and Code of Practice for Research Degree Programmes and that it has not been submitted for any other academic award. Except where indicated by specific reference in the text, the work is the candidate's own work. Work done in collaboration with, or with the assistance of, others, is indicated as such. Any views expressed in the dissertation are those of the author.

SIGNED: Joshua Rees-Garbutt DATE: 30 / 09 / 2020

Table of Contents

List of Figures

List of Tables

Chapter 1 - Introduction

1.1 Statement of Collaboration

Sections of the Introduction have previously been published with the author of this thesis as the lead or co-first author. The publications and contributions are listed below. The sections reproduced here are solely this author's work.

- Landon, S., Rees-Garbutt, J., Marucci, L. & Grierson, C. Genome-driven cell engineering review: in vivo and in silico metabolic and genome engineering. *Essays Biochem.* **63**, 267–284 (2019).
    - Co-first author.
- Rees-Garbutt, J. *et al*. Designing minimal genomes using whole-cell models. *Nat. Commun*. **11**, 836 (2020).
    - Lead author
- Rees-Garbutt, J., Grierson, C. & Marucci, L. Testing theoretical minimal genomes using whole-cell models. *bioRxiv.* doi:10.1101/2020.03.26.010363 (2020).
    - Lead author
- Rees-Garbutt, J. Minesweeper Genome Design Algorithm. *Github*. https://github.com/squishybinary/Minesweeper (2020).
    - Lead author

1.2 Genome Design: Prior State of the Field

Genome engineering is the extensive and intentional genetic modification of a replicating system for a specific purpose [1]. Genome design is the selection of the genetic modifications to produce the specified purpose. Currently, genome engineering progresses by systematic comparison of experimental results, with genome design primarily occurring in the initial stage of experiments. In comparison, metabolic engineering has an established process of cycling between *in-silico* design (100s of models, 10s of algorithms) [2,3] and *in-vivo* editing (100s of strains of several bacterial species) [4]. The designs can be tested *in-silico* before they are implemented in the lab (which is cheaper and faster if the computational infrastructure is available), reducing the quantity of lab experimentation and decreasing the risk of errors [5]. To do this, you

need a model of the cell, an algorithm (analogous to a recipe, a series of steps that solves a problem), and adequate genetic tools to recreate the resulting design in living cells.

The development of whole-cell mathematical models [6] and CRISPR-cas9 gene editing techniques [7–9], could result in a similar combination of experimental biological research and computational mathematical modelling (*in-silico* design and *in-vivo* editing) for genome engineering. At the start of this project, models and genome engineering tools were available, but algorithms for genome design had not been developed.

To create an algorithm we need to have a genome engineering problem to solve. Minimal genomes, cells containing only the DNA essential to survive until successful division, are the best proof-of-concept for genome engineering as they have a simple functionality assessment, the cell either replicates or not [10]. *Mycoplasma genitalium* (*M.genitalium*) is used in minimal genome research as it has the smallest genome of any naturally occurring, self replicating organism. This has made it the basis for minimal gene sets and the first whole-cell model [6], but little other progress has been made with the species as it is difficult to use experimentally [11–13].

1.3 Genome-driven Cell Engineering

Synthetic biology is the rational design and engineering of cells and cellular systems using genetic manipulations [14,15]. It is divided into three fields [16] : DNA-based device construction (production of functioning biological components to be inserted into cells), synthetic protocell development (construction of rudimentary representations of living cells), and genome-driven cell engineering. For more about DNA-based device construction principles see Brophy and Voight [17] and for an introduction to protocell development see Dzieciol and Mann [18]. To introduce this thesis, I will focus on genome-driven cell engineering. Table 1 provides a summary of the key terms in the field.

Genome-driven cell engineering encompasses both metabolic engineering (control of cellular production processes) and genome engineering (production of minimal

genomes, recoded genomes, and cellular chassis/factories). It encompasses diverse types and scales of genetic modifications and underscores the genome as the major driver of cellular events [16]. Metabolic engineering attempts to improve titre, accumulation rate, and yield of a specific metabolite, often from microorganisms in an industrial setting [19]. Genome engineering attempts to generate understanding, reduce risks, and improve metabolite production. Minimal genome research generates understanding by comprehending biology through the engineering of cellular systems [20]. Recoding genomes produces reduced risk by restricting bacteria to specific media and preventing viral transformation [21]. "Optimal" chassis cell development produces improved metabolite production through a variety of means [22].

Genome engineering	Extensive and intentional genetic modification of a replicating system for a specific purpose [1].
Metabolic engineering	Enhances the production of native or introduced metabolites, often in a microbial strain [19].
Minimal genomes	Reduced genomes containing only the genetic material essential for survival, with an appropriately rich medium and no external stresses. No single gene can be removed without loss of viability [13].
Minimal gene sets	Genes selected to produce a minimal genome, but have yet to be tested.
Recoded genomes	Genomes with codon/s that have been freed, substituting codons for synonymous codons that encode the same amino acid, so that they can be assigned to new functions [23,24].
Platform Cell / Cell Factory / Chassis (interchangeable)	A bacterial species that can efficiently convert raw materials into a product of interest, through genome engineering or hosting genetic components [19,25–28].
Multiplex gene editing	Simultaneous introduction of multiple distinct modifications to a genome [29].
Algorithm	Series of steps or rules to attempt to solve a problem, often implemented in a computer.
Model	Mathematical description of a system.
Metabolic Flux	Metabolic reaction rate (i.e. turnover of molecules through a metabolic reaction).
Genome-scale biological models	Category of models containing: metabolic models, transcription regulatory networks, protein–protein interaction networks, integrated cellular models, and whole-cell models [3].
Genome-scale metabolic models	Models representing all active reactions in a cell/organism as a matrix of compound values, and linking reactions with gene products that catalyse them. Abbreviated to GSMMs or GEMs [30,31].
Whole-cell models	Describe the life cycle of a single cell, modelling individual molecules and interactions, and includes the function of every known gene product [6].

Table 1. Key Terms

1.3.1 Metabolic Engineering *in-vivo*

Metabolic engineering uses genetic editing to introduce or modify the desired pathway, while taking control of core metabolism, cellular regulation and stress responses [4,19]. Applications are wide ranging, including fuels, feed additives, and pharmaceuticals [4,32], with the application determining the most appropriate bacteria for production (Table 2).

Industry ready strains only exist for a small number of bacteria such as: *Escherichia coli (E.coli)*, *Bacillus subtilis (B.subtilis)*, *Streptomyces sp.*, *Pseudomonas putida* [4], and *Corynebacterium glutamicum* [19]. Requirements for industrial bacteria are simple nutritional needs, fast and efficient growth, high resistance to extreme physical and chemical conditions, and efficient secretion systems [19]. Also required are sufficient genetic and metabolic knowledge about the species, a range of genetic tools (such as promoters and terminators with varying expression levels), and well-characterised plasmids for precise manipulations. Metabolic engineering has recently been reviewed for *E.coli* [33] and *B.subtilis* [34]. Due to the development of CRISPR-cas9 gene editing tools [35,36], a number of novel bacterial species are becoming available, including *Vibrio natriegens* (with the shortest known doubling time, at 15 minutes), Cyanobacteria (which are photosynthetic) and *Roseobacter* and *Halomonas* (which are marine species with high salt tolerances) [19]. Some examples of organisms used for metabolic production are summarised in Table 2.

The metabolic production pathway is constructed, reconstructed or tweaked in the strain, and can then be iterated upon to produce improvements in titre, rate, and yield. There are six well established strategies [34] for improving these: i) modular pathway engineering, which divides up the production pathway to produce and combine modules with different expression levels [37]; ii) cofactor engineering, in which metabolic flux to the desired products is enhanced through gene edits that alter non-protein cofactor levels [38]; iii) scaffold-guided protein engineering, where the spatial locations of proteins in the cell are modified to increase local concentrations of intermediates [39]; iv) transporter engineering, which improves the import of substrates [40] and export of products [41]; v) dynamic pathway analysis which identifies unknown network interactions and promotes or suppresses them to increase levels of product [42]; and vi) evolutionary

engineering, which mimics natural evolutionary approaches to produce greater amounts of product [43–45].

The development of an "industry ready" strain takes several years and is costly. Strains for Artemisinin and 1,3-propanediol production took 10 years and $50,000,000, and 15 years and $130,000,000 to develop, respectively [4], though sales of metabolic products are expected to reach $6.2 billion dollars by 2020 [46]. The time and cost is due to complex interactions and regulations in metabolism. Metabolite intermediates and products can cause toxicity and act as inhibitors of other reactions, or be misrouted or modified by unrelated enzyme reactions, leading to decreasing titre, rate, and yield [19].

Recently, the availability of accurate genome-scale metabolic models, refined with data captured using omics technologies, has begun to overcome these limitations and support rounds of *in-silico* design and *in-vivo* editing [19,34].

Bacteria	Primary Feature	Applications	Product Examples	Strain Examples
Escherichia coli	Variety of tools / knowledge	Exploratory production, established industrial strain	1-3-Propanediol, 1-4-Butanediol, butanol, insulin, limonene, L-threonine, L-serine, PHAs, propane, succinate.	Based on K-12 and B ancestor strains. Derivatives of MG1655, W3110, BW25113. Specific strains: BL21 Rosetta, DH1, ATCC 31884, DH10B.
Bacillus subtilis	Efficient secretion systems	Protein production	amylases, bacitracin, biotin, cellulosome, chiral stereoisomers, cobalamin, glucanases, guanosine, laccases monophosphate, riboflavin, subtilisin, vitamin B6.	Proteases-defective mutants: WB600, WB800. Specific strains: 168, RH33, BSUL08, 1A1, E8, KU303
Pseudomonas putida	Chemical resistance	Harsh conditions and toxic product production	3-methyl-catechol, anthranilate, cinnamic acid, PHAs, phenol, o-cresol, styrene, terpenoids, vanillate.	Specific strains: KT2440, EM42, Gpo1, S12
Cyanobacteria	Photosynthetic	Light-driven production	1-butanol, 1,3-propanediol, bisabolene, ethanol, farnesene, isoprene, isopropanol, PHAs.	Specific strains: PCC-6803, PCC-7942. PCC-7002

Table 2. A selection of bacteria used for metabolite production. Information collated from Nielsen and Keasling [4], Calero and Nikel [19], Gu *et al.* [34], and Pontrelli *et al.* [33].

1.3.2 Genome Engineering *in-vivo*

Genome engineering is the production of modified genomes using either a prescriptive, existing genome design or a clear laboratory based algorithm to iteratively produce gene edits, along with accurate genetic tools that can be used repeatedly.

1.3.2.a Essentiality

A simple definition of a cell as "living" is if it can reproduce, an *"essential"* gene being indispensable for cell division. A *"non-essential"* gene can be removed without preventing division [10,47]. But a cell's need for specific genes and their products is dependent on the external cellular environment (i.e. how cells are grown [48]) and on the genomic context [10], which is the presence or absence of other genes in the genome and resulting gene products, that can change each time a gene is removed.
Some essential genes can become dispensable with the removal of a particular gene (i.e. a toxic byproduct is no longer produced, so its removal is unnecessary), referred to as "protective essential" genes [10,49,50]. Likewise, some non-essential genes become essential when a functionally equivalent gene is removed, leaving a single pathway to a metabolite (a "redundant essential" gene pair). Gene products can perform together as a complex, with individually non-essential genes involved in producing an essential function [51]. When enough deletions accumulate to disrupt the group, the remaining genes become essential. The cellular death that occurs when redundant essential genes are removed together, or complexes are disrupted, is referred to as synthetic lethality [12,13,52]. Other important classifications for genome engineering include quasi-essential, where removal reduces growth rate substantially [12]; and synthetic rescue, where multiple genes that are essential individually, can be removed together [53,54].

A recent review [10] updates gene essentiality from a binary categorisation to a gradient with four categories: no essentiality (if dispensable in all contexts), low essentiality (if dispensable in some contexts, i.e. redundant essential and complexes), high essentiality (if indispensable in most contexts, i.e. protective essential), and complete essentiality (if indispensable in all contexts). These broad labels describe an individual gene's essentiality in different genomic contexts, and are compatible with other labels that explain underlying mechanisms and interactions in greater levels of detail. Single gene knockout studies (implemented by systematic removal, inactivation, transposon mutagenesis, and antisense RNA [55]) are still used to provide an initial assessment of gene essentiality, but further work is required to assess essentiality dependent on environmental context and genomic context [10].

This redefinition of essentiality has underlined the existence of multiple minimal genomes for individual bacterial species, depending on environmental conditions [10,28], and the selection of redundant genetic pathways in the cell [56].

1.3.2.b Species of Interest

There are several species of interest for genome engineering research: *M.genitalium* due to its size, *E.coli* due to its existing knowledge base, and *JCVI-syn3.0* due to its synthetic nature and size. There are a number of sequenced insect endosymbionts (*Wigglesworthia glossinidia*, *Blochmannia floridanus*, *Buchnera species* etc [57]) that are noted in the literature for their small genome sizes. However, due to their parasitic lifestyle they are not suitable for genome studies and, in addition, most are labelled as *Candidatus* due their inability to be cultured in the lab. *S.cerevisiae*, apart from an entry in Table 3, is not considered in this thesis. Although there is considerable interest and progress being made on synthetic chromosomes in yeast (with a complete design for each chromosome and a third of the chromosomes having been synthesised and assembled [124]), in this thesis I am focused on prokaryotes.

1.3.2.b.1 *Mycoplasma genitalium*

The smallest genome of an independent organism in nature belongs to *M.genitalium*, at 0.58 mb and 525 genes [58]. As a human parasite that has shed functional redundancies over evolutionary time it has proved a useful starting point for comparative genomics. The estimated number of essential genes ranges from: 256 by comparative genomics [59]; 388 by global transposon mutagenesis and comparative genomics [60]; and 381 by single gene knockout [61]. Extrapolating a comparison of single gene deletions of *Mycoplasma* genomes and the genome of *JCVI-syn3.0*, resulted in a prediction of 413 genes for a minimal *Mycoplasma* genome [13]. The number of genes that do not have an annotated function has been reported as 111 [60] and 134 [61]. The *M.genitalium* genome has been reproduced elsewhere, constructed from 25 synthetic parts within yeast [62]. It is also the subject of the first computational whole-cell model [6].

1.3.2.b.2 *JCVI-syn3.0*

JCVI-syn3.0 is a near minimal version of the first synthetic microorganism *JCVI-syn1.0*, reduced by over 50% through hundreds of iterative cycles [12] (originally derived from *Mycoplasma mycoides*). A large amount of interest has been generated post publication, due to its unique synthetic nature, though at publication it contained 149 genes of unknown function (66 genes have subsequently been assigned a function [63]) and has only recently had a metabolic model developed [64]. Recent estimates suggest that of the genes of unknown function an additional 60 genes could be removed to make the genome truly minimal [13].

1.3.2.b.3 *Escherichia coli*

E.coli is the workhorse of microbiology research. Although much larger in size, 4.6 Mbp for K-12 derivative MG1655 (4288 genes, 38% unannotated [65]), it has easy to use and well established methods and techniques, and a Species Knowledge Index 31 times larger than *M.genitalium*'s [28]. Previous research has investigated single gene essentiality in *E.coli* [66], produced single gene deletion libraries for *E.coli* via Tn5 mutagenesis [67] and in-frame knockouts of 3985 genes [68], along with double mutant libraries to investigate synthetic lethality [69]. The number of essential genes has been estimated as 303 by single gene deletion [68] and 630 by transposon based analysis [70], though estimates for reductions that maintain wild type growth are between 2600 and 3000 genes [22,71]. Excitingly, an *E.coli* cell with a synthetic, recoded genome has recently been produced [72] (Section 1.3.2.e).

1.3.2.c Minimal Gene Sets

Genome engineering builds on historical gene essentiality research (Figure 1). The sequencing of small bacterial genomes [58,73] led to comparative genomics, initially between pairs of bacteria [59], then greater numbers of bacteria as genome sequencing increased, which led to the development of minimal gene sets [6,55,57,59–61,74–77].
Minimal gene sets are lists of genes selected to produce a minimal genome, but have yet to be tested. A minimal genome is a reduced genome containing only the genetic material essential for survival, with an appropriately rich medium and no external stresses. No single gene can be removed without loss of viability [13].

Minimal gene sets and minimal genomes focus on protein-coding genes ignoring: essential promoter regions, tRNAs, small noncoding RNAs [28], regulatory noncoding sequences [55], and the physical layout of the genome [55,78]. Predictions for the size of a viable, generic, bacterial minimal genome range from 151 genes [75], to between 300 and 500 genes, though up to 1000 genes if the cell is required to survive on minimal media [78]. As the ratio of genes to base pairs is approximately one gene per kilobase [79], the size expectation of a minimal genome is between 0.15 mb and 0.5 mb.

M.genitalium is the focal point of minimal gene set creation due it is naturally small genome size and available sequenced genome [58]. Minimal gene sets are constructed using three different approaches: protocell development; universal minimal genome theory or comparative genomics; and single gene essentiality research.

1953	DNA Double Helix published (Watson and Crick)
1958	Central Dogma of Biology published (Crick)
1977	Sanger sequencing and first genome sequence published (Sanger et al)
1984	*Mycoplasmas* proposed as models for understanding life (Morowitz)
1990	BLAST (Basic Local Alignment Search Tool) published (Altschul et al)
1995	*Haemophilus influenzae* genome sequenced (Fleischmann et al)
	Mycoplasma genitalium genome sequenced (Fraser et al)
1996	*Saccharomyces cerevisiae* genome sequenced (Goffeau et al)
	Comparison of *Haemophilus influenzae* and *Mycoplasma genitalium* genomes (Mushegian and Koonin)
1997	*Escherichia coli* genome sequenced (Blattner et al)
	Bacillus subtilis genome sequenced (Kunst et al)
1998	Lambda Red homologous recombination (recombineering) proteins inserted into *E.coli* (Murphy)
1999	Global transposon mutagenesis of *Mycoplasma genitalium* (Hutchison et al)
	E-Cell: software environment for whole cell simulation published (Tomita et al)
2000	KEGG: Kyoto encyclopedia of genes and genomes published (Kanehisa and Goto)
2001	*E.coli* minimal genome factory project launched
2004	SB1.0: the first international conference for synthetic biology
2006	*E.coli* single gene knockout library (Keio collection) published (Baba et al)
	E.coli MDS43 (15% reduction) published (Posfai et al)
2007	*Saccharomyces cerevisiae* version 2.0 project launched
	Genome transplantation in *Mycoplasmas* published (Lartigue et al)
	Assembly of *Haemophilus influenzae* genome inside of *E.coli* (Holt et al)
2008	Synthetic *M.genitalium* genome assembled inside of *S.cerevisiae* (Gibson et al)
	E.coli double gene knockout library published (Butland et al)
2009	Gibson Assembly published (Gibson et al)
	Genome transplantation of *Mycoplasma mycoides* into *Mycoplasma capricolum* (Lartigue et al)
	MAGE: multiplex automated genome engineering published (Wang et al)
2010	1000th genome sequenced (Lagesen et al)
	Mycoplasma mycoides JCVI-syn1.0 produced (Gibson et al)
2011	Discovery of tracrRNA in *Streptococcus pyogenes* (Deltcheva et al)
	CAGE: conjugative assembly genome engineering and first genome recoded *E.coli* published (Isaacs et al)
	E.coli Δ33a (38.9% reduction) published (Iwadate et al)
	Synthetic *S.cerevisiae* chromosome arm published (Dymond et al)
2012	Fusion of tracrRNA and crRNA to produce CRISPR-Cas9 guidance system (Jinek et al)
	M.genitalium whole-cell model published (Karr et al)
2013	RNA-guided editing of *E.coli* by combining CRISPR-Cas9 and recombineering published (Jiang et al)
	E.coli DGF-298 (35% reduction) published (Hirokawa et al)
	Direct cell to cell transfer of bacterial genomes into yeast (Karas et al)
2014	MEGA method for genome reduction published (Xue et al)
	Synthetic *S.cerevisiae* full chromosome published (Annaluru et al)
2016	*Mycoplasma mycoides* JCVI-syn3.0 (50% reduction) published (Hutchison et al)
	CasHRA method and *E.coli* MGE-syn1.0 (77% reduction) in *S.cerevisiae* published (Zhou et al)
	CRMAGE: method combining CRISPR-Cas9 and MAGE published (Ronda et al)
	RNA-guided editing of *E.coli* using CRISPR-Cas9 and non-homologous end-joining published (Su et al)
	RNA-guided editing of *Mycoplasma* genome within *S.cerevisiae* using CRISPR-Cas9 (Tsarmpopoulos et al)
	rE.coli-57 (57-codon genome) published (Ostrov et al)
2017	*B.subtilis* PG10 and PS38 (36% reductions) published (Reuß et al)
	Five synthetic *S.cerevisiae* chromosomes published (Richardson et al)
2018	MinGenome: genome design algorithm for genome-scale metabolic models published (Wang and Maranas)
2019	*E.coli* Syn61 (synthetic 61-codon genome) published (Fredens et al)
2020	GAMA + Minesweeper: genome design algorithms for whole-cell models published (Rees and Chalkley et al)

Figure 1. An incomplete history of genome engineering in microorganisms.

Minimal gene sets designed as protocells are not expected to function as full cells, instead functioning as a self-replicating, membrane-encapsulated collection of biomolecules [75], with the sets containing very small numbers of genes.

The universal minimal genome concept is a theory that comparing bacterial genome sequences for common genes will give a gene list that represents essential functions of the cell, and may resemble LUCA (the Last Universal Common Ancestor for life on Earth) [80]. This has been used to construct minimal gene sets, however, as the number of genomes sequenced have increased, the hope around discovering a universal minimal genome strictly from genetic sequences has decreased. Lagesen [81] found that only four genes are recognisably conserved among 1000 bacterial genomes, and even among the evolutionarily reduced *Mycoplasmas* only 196 orthologs (genes that have evolved differently from an ancestral gene but are still recognisably related and retain the same function) were found across the 20 species sequenced [13]. This apparent low conservation of cellular functions is due to non-orthologous gene displacements, independently evolved or diverged proteins that perform the same function but are not recognisably related [13,59]. This means that minimal gene sets designed by the universal minimal genome concept or comparative genomics (subsequently referred to as comparative genomics) could remove a large numbers of genes essential to *M.genitalium* depending on the number of bacterial genomes compared, as those genes are not required by the other bacterial species. Fewer genes may be removed if a smaller number of genomes are compared. This comparative work continues to be built on computationally, analysing the growing number of genomic data sets for key features that could be used to match non-orthologous gene displacements [82].

Minimal gene sets designed using single gene essentiality experiments should, in theory, not remove any essential genes, but do fall prey to issues with transposon mutagenesis, with differing transposon variants, antibiotic resistance genes, and growth periods producing different essentiality classification for genes [83,84].

There are eight minimal gene sets in the literature with detailed gene lists (Table 21). Two are designed as protocells: Tomita *et al.* [74] and Church *et al.* [75]. Three are designed from comparative genomics: Mushegian and Koonin [59], Huang *et al.* [76], and Gil [55,57]. Three are designed from single gene essentiality experiments: Hutchison *et al.* [60], Glass *et al.* [61], and Karr *et al.* [6]. Due to the difficulty of using *M.genitalium* in the lab [85], combined with its long replication time of 12 - 15 hours [11–13], none of these minimal gene sets have been tested as minimal genomes, even with modern techniques [11].

1.3.2.d Minimal Genomes

Previously large-scale gene reductions were either prescriptively designed, with requirements based on existing biological knowledge, or knowledge generated by extensive laboratory testing of the essentiality of individual genes. Most commonly, the reductions were placed into several groups by genome location, with each modified genome segment tested individually in different cells, before being combined into a single cell. This process is, however, time consuming and expensive due to the limitations of current techniques and unexpected cell death caused by unknown genetic interactions. This hinders progress as laboratories can only follow a small number of high-risk research avenues with limited ability to backtrack [10]. The largest scale genome reductions to date (Table 3) include: *JCVI-syn3.0* [12], and *B.subtilis* PG10 and PS38 [86] which were produced to understand and identify minimal genomes; and *E.coli* Δ 33a [87] and DGF-298 [22] which were produced as chassis cells for industrial production.

Research for understanding minimal genomes and developing chassis (Table 4) both involve large numbers of gene/base pair deletions and use similar genetic tools. However, they differ in intent: no single gene can be removed without loss of viability in minimal genomes [13], whereas the cellular growth rate is maintained or promoted in chassis development. Additionally, minimal genomes ignore any cellular component that is not protein-coding genes, at least some of which will be of interest to chassis development. Finally, bacterial species that do not have a use industrially are of use in minimal genome research. *M.genitalium* only synthesizes DNA, RNA, and proteins from imported precursors, in order to replicate itself [13], which it does slowly in a stress-free

laboratory environment [12]; useful for understanding the minimal requirements for life, but not for industry.

Regardless of the original intent, minimal genome reduction strains can have emergent beneficial properties [88,89] in addition to the lower metabolic burden and increased metabolic efficiency produced by reducing gene numbers [90]. The reduced internal biochemistry may also provide benefits as it will interfere less with introduced external pathways [91], making for improved chassis cells. Two minimal genome reduction strains have been subsequently used for production purposes (Table 3).

1.3.2.d.1 Laboratory Techniques

To conduct minimal genome research *in-vivo* you can either: construct bacterial genomes from scratch, and insert them to replace an existing genome, or remove sections of existing natural genomes.

Synthetic genome construction is not currently possible in the majority of bacteria due to economic and technological constraints. Economically, bacterial genome production is too expensive for most institutes. Producing *JCVI-Syn1.0* was estimated to cost ~$40,000,000 [92] and *E.coli* Syn61 [72] cost an estimated $322,000 (409 stretches of 4 - 15 kb of DNA, at 7 cents a base pair). Technologically, megabase sized genomes can be constructed in yeast [62,93], however successful genome transplantation has only been demonstrated in a few *Mycoplasmas* [94–96] and is mutagenic [96]. Genome construction has been completed in *E.coli* [72] in 100 kbp fragments.

1.3.2.d.1.a In *E.coli*

The *E.coli* genome has a long history of being reduced, by: 7% [97], 29.7% [98], 15% [99], 21% [71,88], 35% [22], 23% [100], via a variety of methods, all with the intent of biotechnical applications rather than true minimisation. An exception is Zhou [93] who constructed a minimal genome of *E.coli* (MGE-01) hosted within yeast, which is only 1.03 mb in size, but it has yet to be reinserted back into an *E.coli* cell. In the majority of cases, the *E.coli* reductions displayed normal growth [22,71,97,99,100], with added benefits including: comparable electro-competency to *E.coli* DH10B and better maintenance of unstable

plasmids [99], 1.5 higher cell density [71], or better cell yield [22]. However, the reductions carried out by Hashimoto [98] resulted in unexpected cell shape, nucleoid size, and nucleoid location. The variety of techniques used in these reductions include: restriction enzyme digestion, transposon insertion, cre-meditated recombination, lambda-assisted homologous recombination, and double stranded break induced homologous recombination.

The basis for modern techniques started prior to the millennium, with sequences encoding Lambda proteins (*exo, bet, gam*) being inserted into *E.coli* for the first time, giving a 70 times higher rate of homologous recombination than wild type *E.coli* [101]. This rate of recombination was further improved by removing the methyl-directed mismatch repair (MMR) system, resulting in 25% of grown cells showing the intended mutation [102]. An overview of lambda protein mediated homologous recombination is available [103]. The creation of the modern CRISPR-cas9 technique started with the publication of cRNA and tracrRNA fusion in 2012 [36,104]. Subsequently, the lambda protein system was enhanced by using CRISPR-cas9 as a guidance and counter selection system, resulting in 65% of grown cells carrying the desired mutation [7].

Variations of this system have been produced since: Pyne [105] produced a three plasmid system; Reisch [106] standardised the two plasmid system; Li [107] demonstrated multiplex gene editing (as did Jiang in 2015 [108]) and demonstrated a 100% success rate for some deletions; Zhao [109] produced a single plasmid system; and Zerbini [9] created a more convenient and thoroughly tested two plasmid system. These CRISPR-cas9 - lambda protein plasmid systems offer many advantages compared to previous techniques including: single step procedures, reusable plasmids, elimination of selection steps, absence of scar sites, and greater success rates across prokaryotes. Zerbini [9] targeted 81 genes using a CRISPR-cas9 - lambda protein plasmid system with double stranded donor DNA. 100% of the genes targeted were successfully mutated, with 10-100% of the colonies grown carrying the mutation depending on the gene targeted for deletion. Due to the plasmid systems being "variations-of-the-theme" these robustness experiments should be applicable across techniques.

A different technique using CRISPR-cas9 was demonstrated by Standage-Beier [110], who used single strand CRISPR-cas9 nicking to guide homologous recombination. There are also CRISPR-cas9 enhanced alternatives to homologous recombination. Su [111] reintroduced non homologous end joining into *E.coli* through inserted *Mycoplasma* genes, in combination with CRISPR-cas9. Zheng [112] later improved this with a different *Mycoplasma* species insert, increasing the maximum deletion size from 17,000 base pairs to 123,000 base pairs. Zhou [93] established a new technique for assembling megabase-sized genomes in yeast, taking advantage of CRISPR-cas9.

1.3.2.d.1.b In *Mycoplasmas*

The development of *Mycoplasma* techniques is in reality the development of yeast hosted genome editing, as this is instrumental for editing organisms that are difficult to culture [11], which includes *Mycoplasmas*. This began by demonstrating the assembly of a *M.genitalium* genome within yeast [62] and the subsequent insertion of a yeast assembled *Mycoplasma* genome into a new *Mycoplasma* cell [11,95]. This has been subsequently improved by the development of direct cell-to-cell transfer of bacterial genomes into yeast [113], and the addition of CRISPR-cas9 within yeast for *Mycoplasma* modification [114].

1.3.2.e Genome Recoding

Genome recoding research substitutes synonymous codons, encoding the same amino acid, across an entire genome resulting in: virus resistance, as viral replication relies on all 64 codons [23]; prevention of gene transfer [115]; and increased translation efficiency [21]. It also produces a blank codon that can be repurposed for a novel function not commonly found in nature such as a non-standard amino acid (NSAA) [21,23,116]. This incorporation of NSAAs additionally acts as a form of biocontainment as the organism is engineered to be dependent upon the presence of the synthetic NSAA to survive.

Genome recoding is possible due to the development of new technologies: MAGE, multiplex automated genome engineering [117]; CAGE, conjugative assembly genome engineering [21,118]; REXER, replicon excision for enhanced genome engineering through programmed recombination [119]; and GENESIS, genome stepwise interchange synthesis

[72]. MAGE cyclically targets many genetic locations to conduct mismatches, insertions, deletions in a single cell or across a population of cells, maintaining high efficiency up to 10 targets at a time [117]. This leads to rapid and continuous generation of genetic diversity for strain and pathway engineering. CAGE is a complementary method to assemble modified genomic modules from individual cells into a single genome through cell to cell transfer, and has been used in combination with MAGE to systematically recode codons [21,118]. REXER inserts pieces of synthetic *E.coli* DNA into the chromosome (within the *E.coli* DNA insertion size limitations) using CRISPR-cas9 to cut both the plasmid and the chromosome. The GENESIS methodology is the use of REXER for sequential stepwise deletions, allowing the entire genome to be replaced without using genome transfer. This development improves the outlook of building minimal genomes *in-vivo* in *E.coli*.

Genome Reductions		
Microbe	Reduction	Benefits
JCVI-syn3.0 [12]	50%	Smallest genome of any autonomously replicating cell. Has a doubling time of ~180 min, four - five times faster than *M.genitalium* (12 - 15 hours [13]).
E.coli Δ33a [87]	39%	-
E.coli DGF-298 [22]	35%	Better growth fitness and cell yield in a rich medium, than the wild type strain, and has a more stable genome.
B.subtilis PG10 and PS38 [86]	36%	Subsequently used for production purposes, as has traits that are favorable for producing 'difficult-to-produce proteins' by overcoming several bottlenecks (secretion process and unstable product) [120].
E.coli Δ16 [98]	30%	-
B. subtilis MGIM [121]	24%	Little reduction in growth rate and comparable enzyme productivity.
E. coli MGF-01 [88]	22%	Better growth rate resulting in 1.5-fold cell density and 2.4-fold greater threonine production compared to the wild type strain.
B. subtilis MBG874 [122]	20%	Extracellular cellulase and protease production were 1.7 and 2.5-fold higher. Production period was elongated and carbon utilisation improved.
E.coli MS56 [100]	23 %	Insertion sequence free, making it more genomically stable, predicted to increase production of recombinant proteins.
E. coli MDS43 [99]	15%	Showed genome stabilization and increased electroporation efficiency, comparable to *E.coli* DH10B. Subsequently used for production purposes: 83% increase in L-threonine production, compared to *E. coli* MG1655 using the same metabolic engineering [90].
Genome Recoding		
Microbe	Modifications	
32 *E.coli* strains [21]	Replaced 314 UAG (stop) codons with UAA.	
E.coli MG1655 [24]	Replaced 321 UAG (stop) codons with UAA.	
r*E.coli*-57 [116]	Replaced 62,214 instances of seven codons (UAG (stop), AGG and AGA (Arg), AGC and AGU (Ser), UUG and UUA (Leu)).	
E.coli C123 [123]	Replaced 123 rare AGA and AGG (Arg) codons from essential genes with 110 CGU conversions and 13 optimized codon substitutions.	
E.coli MDS42 [119]	Tested 1468 codon changes using REXER plasmids and GENESIS method.	
S.cerevisiae Sc2.0 [124]	Replaced all UAG (stop) codons with UAA.	
E.coli Syn61 [72]	Replaced 18,214 codons, UCG with AGC, UCA with AGU, UAG with UAA, using REXER plasmids and GENESIS method.	

Table 3. Genome-driven cell engineering examples.

Combining genome engineering research efforts together can give insights into what an "optimal" cellular chassis could look like (Table 4) and suggest research pathways going forward.

Feature	Description
Genetically Stable	Removal of mobile DNA elements (e.g. insertion elements, transposases, phages, integrases, site-specific recombinases) [125].
Genomically Recoded	Substitute codons to create blank codons for inclusion of new, non-natural amino acids [21], decreased likelihood of viral infection [23], and horizontal gene transfer [115].
Genome Minimised	Removal of competing and unwanted metabolic pathways that divert the resources of the cell away from desired end products [1], resulting in increased capacity for and reduced impact of cellular burden [126,127], and greater robustness and energy efficiency [128]. Reduced transcriptional regulatory interactions also result in lower resistance to engineering efforts [125]. Additionally, creates larger and more optimal precursor pools [129].
Efficient Production	Simple nutritional needs, fast and efficient growth, and efficient secretion systems [19].
Robust	Tolerance for extreme conditions [19] i.e. strength of cell membrane or wall and appropriate coping mechanisms [28].
Well Understood	Sufficient knowledge of the organism's genome and metabolism to produce accurate models, and allow modularisation of metabolic pathways [28].
Developed Tools	A range of established genetic tools for manipulation, including promoters and terminators with varying expression levels, and well-characterised plasmids, to enable titre, rate, and yield improvements and rapid and efficient tuning of genetic components [1].

Table 4. Features of an optimal chassis for a wide range of applications.

1.3.3 Metabolic Engineering *in-silico*

Models of a cell's metabolism (known in the literature as genome-scale metabolic models, abbreviated to GSMMs) are constructed from metabolic networks, which consist of all known metabolic reactions and the genes which encode each enzyme. These models produce a table of reactions, substrates, and biologically feasible constraints from the metabolic network, and form a solution space (all the possible outputs given all the possible inputs) by analysing each reaction in the model. An algorithm or optimisation process (such as flux balance analysis (FBA), see Orth *et al.* [130] for a recent review) can be applied to the model to find a solution to a defined problem, by modifying the rate of reactions in the model.

University of California San Diego maintain a database of published GSMMs which have been validated against experimental data. As of the most recent update (February

2018) there are 113 bacteria, 57 eukaryote and 8 archaea manually curated models [131]. GSMMs can also be generated automatically starting from available biological databases (e.g. KEGG [132]) or generic universal networks that are subsequently modified. Automation tools include: modelSEED [133], Pathway Tools [134], AuReMe [135], Merlin [136], MetaDraft [137], Raven [138], and CarveMe [139], which have been recently reviewed [140].

Metabolic models can be used to predict single gene essentiality, using the lack of biomass production as a proxy for cellular death, demonstrated with *E.coli* MG1655 (86% accuracy) [141], *B.subtilis* [142], *Saccharomyces cerevisiae* [143], *Helicobacter pylori* [144], and *Corynebacterium glutamicum* [145].

Although referred to as genome-scale, these models only account for the metabolism and no other systems within the cell. Recently developed models have begun to extend metabolic models to include transcription and translation reactions [146]. Metabolic models might not be appropriate for genome engineering as they cannot predict gene essentiality in the genomic context of the entire cell (Section 1.3.2.a), with the potential to label genes that are essential for the cells correct function outside metabolism incorrectly.

1.3.4 Genome engineering *in-silico*

The *M.genitalium* whole-cell model [6] was, at the time of publication, the only existing model that included the function of every known gene product (401 of the 525 *M.genitalium* genes), making it capable of modelling genes in their genomic context (Section 1.3.2.a).

A single *in-silico M.genitalium* cell is simulated from random, biologically feasible initial conditions until the cell divides or reaches a set time limit. The model (implemented in MATLAB) combines 28 cellular sub-models, with parameters from >900 publications and >1,900 experimental observations, resulting in 79% accuracy for single gene knockout essentiality [6].

The whole-cell model is similar to metabolic models, with reactions and biological constraints implemented as equations and variables, however in the whole-cell model the equations are grouped into 28 cellular sub-models (which are modelled independently using different mathematics and experimental data) and the variables are grouped into 16 cellular states. The whole-cell model integrates the 28 cellular sub-models using time, which consists of:

- When the model is first run, the variables in the 16 cellular states are initialised and allocated among the 28 sub-models.
- The cellular sub-models then operate for one second.
- The variables in the 16 cellular states are updated and re-allocated to the 28 sub-models.
- This repeats until the cell divides, or the simulation time reaches a maximum value (13.89 hours).

There are six categories of the 28 cellular sub-models: transport and metabolism, DNA replication and maintenance, RNA synthesis and maturation, protein synthesis and maturation, cytokinesis, and host interaction (more detail is provided in Table 5).

Outside of single gene knockout simulations, it has been used to investigate discrepancies between the model and real-world measurements [147], design synthetic genetic circuits in the context of the cell [148], and make predictions about the use of existing antibiotics against new targets [149].

Development of whole-cell models for genome engineering is time and cost intensive. The *M.genitalium* whole-cell model took ten-person years to build [150], resulting in the Karr Lab, Mount Sinai developing automation tools (Datanator and WC-Lang [151]) along the lines of automated tools for producing GSMMs.

Grouping	Sub-model	Description	Mathematical Modelling	Limitations
Transport and Metabolism	Metabolism	Modelled nutrient import, conversion and recycling of metabolic building blocks. The interface between the environment and the other 27 sub-models.	FBA, trained using the observed growth rate of *M.genitalium*	Reconstructed by matching *M.genitalium* to *E.coli* genes, using a comprehensive *E.coli* metabolic model. Used observed chemical compositions from other species, *M.gallisepticum* and *E.coli*.
RNA Synthesis and Degradation	Transcription	Models the state of each RNA polymerase, initiation at promoters, elongation, NTP allocation, and termination.	Markov chain (RNA polymerase), trained using *M.genitalium* observations. Stochastic process (initiation), trained using reconstructed expression and decay rates.	Chromosome organisation reconstructed from *M.pneumoniae*. Transcription termination not well-characterised, modelled deterministically, proceeding based on presence of copies of transcription termination factors.
RNA Synthesis and Degradation	Transcriptional Regulation	Models the binding of transcriptional regulators to promoters, and the effect (fold change) on subsequent RNA polymerase binding.	Algorithm, with stochastic processes, that runs every time step. Trained using reported regulator fold change effects.	Not well characterised in *M.genitalium*. Reconstructed using DBTBS database. Assumptions: Transcriptional regulators bind promoters proportionally to their fold change effect; only one copy of each regulator can bind at each promoter; stable binding with DNA (only displaced by other DNA-binding proteins); regulator impacts multiply as additional regulators bind.
RNA Synthesis and Degradation	RNA Processing	Models the cleavage of polycistronic, non-coding (operonic) RNA.	Algorithm, with mass-action kinetics, that runs every time step. Expression was fit to prevent sustained accumulation of unprocessed RNA.	Reconstructed based on *E.coli* RNA cleavages linked to *M.genitalium* enzymes. Assumptions: cleavage occurs in a single reaction, in a single time step; and rate is determined by copy numbers of RNA, metabolite, and enzyme.
RNA Synthesis and Degradation	RNA Modification	Models the base-specific modification of non-coding RNAs, to encode the triplet codes, assist folding, and stabilise products.	Algorithm, with mass-action kinetics, that runs every time step. Expression was fit to prevent sustained accumulation of unprocessed RNA.	Reconstructed based on *E.coli* RNA modifications. Assumptions: cleavage occurs in a single reaction, in a single time step; and rate is determined by copy numbers of RNA, metabolite, and enzyme.
RNA Synthesis and Degradation	tRNA Aminoacylation	Models the aminoacylation of tRNAs.	Two step process: 1) calculate the maximum number of reactions, 2) randomly select and conduct a reaction. Restart and recalculate Step 1, until resources are exhausted.	Assumptions: Intermediate steps are not represented. Reactions are randomly selected using a probability distribution, weighted by maximum number of reactions and what has already occurred.
RNA Synthesis and Degradation	RNA Decay	Models the decay of RNA, recycling the small RNA pool using ribonuclease R.	5 step process, using first-order Poisson process (rate is known, occurrence is random), dependent on ribonuclease R counts, and selection based on RNA half-life.	Uses observed decay rates from *E.coli*. Assumptions: RNA degradation is an all-or-nothing event that proceeds to completion in a single timestep.
Protein Synthesis and Degradation	Translation	Models the assembly of 70S ribosomes, polymerisation, allocation of tRNAs to ribosomes, and termination of stalled ribosomes.	Three processes that randomly select targets. Initiation: dependent on free ribosomes, initiation factors, and energy. Elongation: dependent on copies of three elongation factors. Termination: dependent on copies of release factor, recycling factor, elongation factor, and energy.	Assumptions: ribosome stalling probability is uncharacterised, so is set to very low. Stalling is a fourth process that is only triggered by a non-advancing elongating ribosome.
Protein Synthesis and Degradation	Protein Processing	Models the first steps in protein maturation.	Algorithm, with stochastic processes, that runs every time step, until resources are exhausted. Expression was fit to prevent sustained accumulation of unprocessed protein.	Some reconstruction based on data from *Shewanella oneidensis*. Assumptions: protein cleaved in a single time step, in a single reaction; and rate is determined by copy numbers of protein, metabolite, and enzyme.
Protein Synthesis and Degradation	Protein Translocation	Models translocation across and in the cell membrane.	Algorithm, with stochastic processes, that runs every time step, until resources are exhausted. Expression was fit to prevent sustained accumulation of untranslocated protein.	Assumptions: translocation doesn't begin until after termination and processing; occurs in a single time step, in a single reaction; and rate is determined by copy numbers of protein, metabolite, and enzyme.

Grouping	Sub-model	Description	Mathematical Modelling	Limitations
Protein Synthesis and Degradation	Protein Processing 2	Models lipoprotein adduction and lipoprotein cleavage.	Algorithm, with stochastic processes, that runs every time step, until resources are exhausted. Expression was fit to prevent sustained accumulation of unprocessed protein.	Assumptions: occurs in a single time step, in a single reaction; and rate is determined by copy numbers of protein, metabolite, and enzyme.
Protein Synthesis and Degradation	Protein Folding	Models folding into 3D structures.	Algorithm, with stochastic processes, that runs every time step, until resources are exhausted. Expression was fit to prevent sustained accumulation of unfolded protein.	Reconstructed based on proteome studies of *E.coli* and *B.subtilis*. Assumptions: As not well understood, the 3D configuration is represented as a two-state – folded, unfolded – Boolean variable; and rate is determined by copy numbers of protein, metabolite, and enzyme.
Protein Synthesis and Degradation	Protein Modification	Models the modification of specific amino acids, regulating structural diversity, activity and expression.	Algorithm, with stochastic processes, that runs every time step, until resources are exhausted. Expression was fit to prevent sustained accumulation of unmodified protein.	Based on observations from *M.genitalium* and *M.pneumoniae*. Assumptions: occurs in a single time step, in a single reaction; and rate is determined by copy numbers of protein, metabolite, and enzyme.
Protein Synthesis and Degradation	Macro-molecular Complexation	Models the formation of protein complexes.	Algorithm, with mass-action kinetics, that runs every time step, until resources are exhausted.	Assumptions: as poorly understood, complexes are assumed to form spontaneously, without assistance; completes in a single time step; each complex forms with the same specific rate; and by simultaneous collision of each sub-unit.
Protein Synthesis and Degradation	Ribosome Assembly	Models the assembly of 30S and 50S ribosome particles	Algorithm, with stochastic processes, that runs every time step, until resources are exhausted. Expression was fit to prevent sustained amino acid accumulation.	Assumptions: As not well understood, the sub-model only represents rRNA transcripts, protein monomers and ribosomal particles; completes in a single time step; and rate is determined by copy numbers of RNA, protein, metabolite, and enzyme (GTP and GTPase).
Protein Synthesis and Degradation	Terminal Organelle Assembly	Models the hierarchical assembly of the eight terminal organelle proteins.	Algorithm, using a Boolean, hierarchical, assembly model.	-
Protein Synthesis and Degradation	Protein Decay	Models the cleavage of proteins and protein misfolding and refolding.	Algorithm controlling multiple processes. Cleavage: uses first-order Poisson process (rate is known, occurrence is random). Misfolding: uses a stochastic process. Refolding: uses a deterministic, single Boolean rule. Expression was fit to prevent sustained accumulation of damaged proteins.	Protein half-life was predicted using N-end rule. Assumptions: membrane and extracellular proteins are not vulnerable to degradation; degradation occurs in a single step; occurs immediately if proteases and energy available; if energy or enzyme limited proteins are degraded stochastically; cystol-localised proteins are refolded; proteins misfold and refold at the same rate, with no intermediates.
Protein Synthesis and Degradation	Protein Activation	Models the chemical regulation of protein activity.	Boolean network of the effects of small molecules, temperature, and pH.	Reconstructed from the DrugBank database. Assumptions: proteins with no known chemical regulation are unaffected.
DNA Replication and Maintenance	Replication Initiation	Models the binding of DNA polymerase to oriC, starting chromosome replication	Algorithm, with deterministic and stochastic steps, for seven stages.	Reconstructed from *E.coli*'s *DnaA* process. All rate constructs were obtained from previous models.
DNA Replication and Maintenance	Replication	Models bidirectional DNA replication following initiation.	Algorithm, with eight deterministic stages, evaluated in a random order to allocate shared resources fairly.	Exact mechanism is unknown. Polymerisation is limited by the average rate in *Mycoplasmas*. Assumptions: RNA polymerase is displaced upon collision with DNA polymerase, but will only stall DNA polymerase for a limited time if a head-on collision.
DNA Replication and Maintenance	Chromosome Segregation	Models chromosome segregation post replication.	Immediate segregation event, under four conditions being met (Boolean). Checked at each timestep.	Poorly characterised, as *M.genitalium* has much fewer proteins than required in other species

Grouping	Sub-model	Description	Mathematical Modelling	Limitations
DNA Replication and Maintenance	Chromosome Condensation	Models chromosome compaction.	Two step process: 1) calculate the maximum number of SMC complexes that can bind, 2) randomly select binding sites using a weighted probability.	Assumptions: model SMC condensation separately from DNA Supercoiling, as unknown impact of SMC activity on DNA linking number (used by supercoiling); SMC complexes bind to random locations; and rate is determined by copy numbers of SMC complexes and energy.
DNA Replication and Maintenance	DNA Supercoiling	Models chromosome compaction also.	Algorithm, with stochastic or calculation steps, for seven stages,, parameterised by observed kinetics.	-
DNA Replication and Maintenance	DNA Damage	Models spontaneous and radiation induced damage.	Reactions are triggered randomly at random locations (each reaction treated as an independent Poisson process), parameterised by reported data.	Spontaneous damage is calculated from the observed rate, whereas radiation damage is calculated from the observed rate and modelled radiation flux. Chemically-induced damage was reconstructed but not modelled (as *M.genitalium* is not commonly cultured with chemically damaging agents).
DNA Replication and Maintenance	DNA Repair	Models five modes of DNA repair, one specific to *M.genitalium*, and the binding of *DisA* to DNA lesions.	Damaged locations randomly selected, independent repair reactions randomly chosen, parameterised by observed kinetic rates, until resources are exhausted.	Assumptions: each step is a separate reaction and independent; the rate is determined by DNA configuration and copy numbers of metabolites and enzymes
Cytokinesis	FtsZ Polymerisation	Models the formation of FtsZ rings.	Ordinary differential equations, evaluated at each time step.	Equations taken from the literature, and simplified. Assumptions: length of filament assumed to be equivalent of *C.crescentus*, and all filaments are of the same length.
Cytokinesis	Cytokinesis	Models the contraction of successively smaller FtsZ rings.	Algorithm, with stochastic processes, for five stages, parameterised by observed rates.	Uses published iterative pinching model, subsequently modified. Assumptions: rates of FtsZ filaments binding and dissociating are uncharacterised, so set to rapid rates that would not limit progress; assume FtsZ ring in *M.genitalium* is 3 filaments thick.
Host Interaction	Host Interaction	Models the interaction of *M.genitalium* with the host human urogenital epithelium.	Algorithm, with four sequential and dependent conditional statements (Boolean).	Reconstructed from observed terminal organelle protein compositions and reported host interactions. Poorly characterised.

Table 5. 28 sub-models that comprise the *M.genitalium* whole-cell model. The inputs and outputs of each sub-model are: metabolite and macromolecule numbers; RNA, protein and DNA polymers configurations; and enzyme configurations and capacity.

Of the prior publications using the *M.genitalium* whole-cell model, one in particular (Sanghvi *et al*. 2013 [147]) is deserving of more discussion. The paper itself is error-prone, and overemphasises its results and implications; but the supplementary results are remarkable. This supplementary material, by itself, is inspirational for the analysis conducted in this thesis; it elaborates on the model implementation, the description of biological processes, and the biological implications. This analysis is the precursor to the analysis conducted in Chapter 4 and 5.

The methodology for using the *M.genitalium* whole-cell model presented in the paper is good: predict cellular behaviour *in-silico*, compare to *in-vivo*, investigate the discrepancies to make discoveries, and update the model.

However, this is only conducted in a very limited manner in this paper. This paper claims to investigate 86 single gene knockout simulations quantitatively. This number is actually 53, due to 34 of the included genes being unmodelled; both this eand the model publication paper omit any quantitative references to unmodelled genes. This paper references “seven categories … two of which (“annotation insufficient” and “quantitative agreement, annotation insufficient”) are dependent on ... whether a gene’s function was well-enough annotated for functional inclusion in the model”. It is not clarified in the paper how, if a gene has an insufficient annotation to be modelled it is capable of yielding quantitative information via *in-silico* simulation. The 34 genes in the category are erroneously included in the count of the 86 single gene knockout simulations (though 86 itself is a miscount, the inclusion of the 34 genes makes it 87).

It focuses on two of these seven categories to select 14 genes to have any, albeit brief, secondary analysis. The paper, however, should consider 28 genes at this stage. The paper miscounts one category (13 instead of 14), updates the results of 9 genes from the model’s publication (Sanghvi *et al*. Supplementary Table 1 compared to Karr *et al.* Supplementary Table S2G [6]) without highlighting this or giving an explanation as to why (reducing a category from 14 genes to 5 genes), and excludes the genes MG_011, MG_012, MG_293, MG_411, MG_412 from analysis arbitrarily. Of these 14 genes, only 3 single gene knockout simulations are investigated in-depth.

The paper proposes a second, good methodology: look for a possible mechanism for the model's inability to predict the experimental data, by examining the 'molecular pathology' in both the model and the literature, thereby identifying a previously misrepresented or missing function in the model.

However, the focus of the paper from here is disappointing. It focuses on the three single gene knockouts due to the quantitative comparisons it can make rather than qualitative (the alternative, qualitative comparisons are what are so compelling in the supplementary results). This would be understandable from the perspective of strengthening the research with laboratory results, except that these experiments occur as plasmid expressions in *E.coli* (which does not bolster the results in the same way).

The quantitative focus results in tweaking parameter values to change metabolic fluxes in identified compensatory reactions, to produce experimentally predicted growth rates in the *M.genitalium* whole-cell model. This is easier to implement in the model but dodges really questioning the model's implementation of biological processes. It also allows the blame for these irregularities to be placed on the lack of *M.genitalium in-vivo* data (as explained in the paper, the prior implementation relied on data "approximated from different organisms"), rather than decisions made by the model developers. Given this is a paper by the same group, a year later, there was little to lose by confronting and improving their own work. Otherwise, the biological explanation given for MG_039 gets the closest to the supplementary result's analysis and is, therefore, the more satisfactory.

The overemphasis appears twice in the paper. First, the paper merges its quantitative results with the previously published qualitative results from the model's publication (for "all 525" genes - as previously stated, 124 are unmodelled), in an unqualified manner. This enhances the scale of the results, which they state more truthfully later is only "three instances of validation". Secondly, the paper states that the results were predictive. As the paper could have more simply adjusted and trialed the three gene's catalytic efficiency, given the known quantitative error with the experimental results, the

use of the model to predict this efficiency is unnecessary and overstates the impact of this result.

This thesis differs from Sanghvi *et al*. 2013 [147] by focusing on qualitative rather than quantitative analysis, increasing the scale of the genetic modifications past single gene knockouts, and conducting thorough analysis across the largest breadth of results for as long as possible; while emulating the analysis conducted in the supplementary results.

1.3.4.a *E.coli in-silico*

The *E.coli* whole-cell model is the second whole-cell model, under review for publication, with its code publicly available [152]. Currently, it has 1214 genes modelled, with the ability to simulate multiple environments and growth conditions, and match against experimentally verified results and predictions. The model is implemented in Python and uses FireWorks, a workflow management tool, to organise and conduct simulations. In comparison to the *M.genitalium* whole-cell model, the *E.coli* whole-cell model is quicker (15 minutes to simulate a life cycle of a cell), can simulate multiple generations, accounts for 50 times more molecules, and is built using species specific data (from primary literature and databases (e.g. EcoCyc)). Most of this data comes from three strains of *E.coli*: K-12 MG1655, B/r, and BW25113, making the model a composite strain.

In more detail, the *E.coli* whole-cell can simulate cells growing at different rates in response to changes in the simulated environment, regulate transcription more accurately by including the function of 22 transcription factors (that regulate 355 genes), model 340 metabolic reactions in greater detail, generate metabolites according to resource availability rather than producing metabolites in a fixed ratio every time the cellular states are updated, and has greater detail in the translation sub-model (translational efficiency data is included to inform ribosome binding to mRNA transcripts) and RNA decay sub-model (decay rates for both endonuclease-mediated cleavage and digestion by exoRNases are included). However, the *E.coli* whole-cell does not currently model all *E.coli* genes and lacks several sub-models implemented in

the *M.genitalium* whole-cell model. These missing sub-models can be grouped into: DNA (chromosome condensation and segregation, damage, repair, supercoiling); RNA (processing, modification, aminoacylation); protein (processing, translocation, folding, modification); ribosome assembly and FtsZ polymerization. The model aims to support simulating 17 environmental conditions, though only five have been fully implemented and tested (wild type, nutrientTimeSeries, condition, metabolism_kinetic_objective_weight, and param_sensitivity), with gene_knockout having been implemented but not tested. Other environmental conditions and increased numbers of modelled genes are planned to be added to the model over time. Greater detail on the model will be made available upon publication.

1.3.4.b Algorithms for Genome Engineering

Genome design algorithms for genome engineering are rare; only one algorithm has been identified in the literature outside of this thesis. MinGenome [153], published in 2018, identifies all dispensable contiguous sequences (unbroken stretches of non-essential genes) in the *E.coli* MG1655 genome. By identifying these large multi-gene deletions and comparing them to existing *E.coli* genome reductions, new deletions to test *in-vivo* are highlighted, which when tested will either increase the current genome reductions or highlight low essential genes.

1.3.5 Issues

There is a need for greater species-specific understanding of the metabolism and the genome. Even well-studied organisms (*B.subtilis* and *E.coli*) have genes with unknown functions and essentiality; bacterial genomes have on average 33% genes of unknown function [154]. Of the genes with known functions, in most cases we only understand essentiality at the single or double gene knockout level [68,69]. Current genome reductions have had to identify synthetic lethal interactions (particularly low essential genes) as part of their reduction efforts, rather than being able to design around them. If we had a greater grasp of gene product interactions, enabling them to be accurately modelled, this could be avoided. We would also be taking steps towards a proposed end goal of genome design, combining modular components of different bacteria in a novel cell [26,155].

1.3.6 Next Steps

The next steps for genome engineering are: (i) the production and publication of new whole-cell models [156]; (ii) the implementation of computational standards to keep the field cohesive and prevent fragmentation [157]; (iii) the testing of *in-silico* designs *in-vivo* [56]; (iv) and the establishment of routine procedures for *in-vivo* genome reductions for species that will soon have whole-cell models.

1.4 PhD Aims

Firstly, to create a genome design algorithm for use with whole-cell models. Secondly, to produce a computational minimal genome for *M.genitalium*, using the *M.genitalium* whole-cell model, the developed genome design algorithm, and the University of Bristol's supercomputers. Thirdly, to test existing and currently untested minimal gene sets *in-silico* using the *M.genitalium* whole-cell model. Fourthly, produce a computational minimal genome for *E.coli*, using the developed genome design algorithm and the *E.coli* whole-cell model when it is made available to outside researchers. Finally, to test large-scale reductions of the *E.coli* genome as predicted by the *E.coli* whole-cell model. This final step would be the first research to test whole-cell model results *in-vivo*, and the first use of *in-silico* design and *in-vivo* editing for entire genome engineering.

Chapter 2 - Methods

2.1 Statement of Collaboration

Sections of the Methods chapter have previously been published with the author of this thesis as the lead author. The sections reproduced here are solely this author's work.

- Rees-Garbutt, J. *et al*. Designing minimal genomes using whole-cell models. *Nat. Commun*. **11**, 836 (2020).
    - Lead author
- Rees-Garbutt, J., Grierson, C. & Marucci, L. Testing theoretical minimal genomes using whole-cell models. *bioRxiv.* doi:10.1101/2020.03.26.010363 (2020).
    - Lead author
- Rees-Garbutt, J. Minesweeper Genome Design Algorithm. *Github*. https://github.com/squishybinary/Minesweeper (2020).
    - Lead author

2.2 General Methods

2.2.1 Code Availability

All code created as part of this thesis is available on Github (github.com/squishybinary, github.com/GriersonMaruccciLab) under a GNU General Public License v3.0 (gpl-3.0). For more information see choosealicense.com/licenses/lgpl-3.0/.

2.2.2 Model Availability

The *M.genitalium* whole-cell model is freely available: github.com/CovertLab/WholeCell. The model requires a single CPU and can be run with 8 GB of RAM. I run the *M.genitalium* whole-cell model on Bristol's supercomputers using MATLAB R2013b, with the model's standard settings. However, I use my own version of the SimulationRunner.m. MGGRunner.m (github.com/GriersonMaruccciLab/Analysis_Code_for_Mycoplasma_genitalium_whole-cell_model) is designed for use with supercomputers that start hundreds of simulations simultaneously. It artificially increments the starting time-date value for each simulation,

as this value is subsequently used to create the initial conditions of the simulation. My research copy of the whole-cell model was downloaded 10th January 2017.

The *E.coli* whole-cell model is freely available: github.com/CovertLab/WholeCellEcoliRelease. I run the model on Bristol's BlueCrystal supercomputer using the standard installation requirements and settings. My research copy of the model is up to date with the current release snapshot, which was released on the 29th October 2019.

2.2.2.a *M.genitalium in-silico* Environmental Conditions

M.genitalium is grown *in-vivo* on SP4 media. The *in-silico* media composition is based on the experimentally characterized composition, with additional essential molecules added (nucleobases, gases, polyamines, vitamins, and ions) in reported amounts to support *in-silico* cellular growth. Additionally, the *M.genitalium* whole-cell model represents 10 external stimuli including temperature, several types of radiation, and three stress conditions. For more information see Karr *et al.* Supplementary Tables S3F, S3H, S3R [6].

2.2.3 Equipment

For the *M.genitalium* whole-cell model I used the University of Bristol Advanced Computing Research Centres's BlueGem, a 900-core supercomputer, which uses the Slurm queuing system, to run whole-cell model simulations.

For the *E.coli* whole-cell model, I used the University of Bristol Advanced Computing Research Centres's BlueCrystal, a 3568-core supercomputer, which uses the PBS queuing system, to run whole-cell model simulations.

I used a standard office desktop computer, with 8 GB of ram, to write new code, and interact with the supercomputer. I used the following GUI software on Windows 7: Notepad++ for code editing, Putty (ssh software) for terminal access to the supercomputer, FileZilla (ftp software) to move files in bulk to and from the supercomputer, and PyCharm (IDE software) as an inbuilt desktop terminal and for

python debugging. The command line software used included: VIM for code editing, and SSH, Rsync, and Bash for communication and file transfer with the supercomputers.

2.2.4 Data Format

For the *M.genitalium* whole-cell model the majority of output files are state-NNN.mat files (Figure 2), which are logs of the simulation split into 100-second segments. The data within a state-NNN.mat file is organised into the 16 cellular variables. These are typically arranged as 3-dimensional matrices or time series, which are flattened to conduct analysis. The other file types contain summaries of data spanning the simulation. Each wild type simulation consists of 300 files requiring 0.3 GB. Each gene manipulated simulation can consist of up to 500 files requiring between 0.4 GB and 0.9 GB. Each simulation takes 5 to 12 hours to complete in real time, 7 - 13.89 hours in simulated time.

The *E.coli* whole-cell model outputs data in plain text files and automatically plots a range of graphs from the data, in .png, .pdf, and .svg formats.

2.2.5 Data Analysis Process

For the *M.genitalium* whole-cell model, the raw data is automatically processed as the simulation ends. runGraphs.m carries out the initial analysis, while compareGraphs.m overlays the output on collated graphs of 200 unmodified *M.genitalium* simulations. Both outputs are saved as MATLAB .fig and .pdfs, though the .pdf files were the sole files analysed. The raw .mat files were stored in case of further investigation.
For the single gene knockout simulations produced, the non-essential simulations were automatically classified and the essential simulations flagged. Each simulation was investigated manually and given a phenotype using the decision tree (Figure 6).

For the *E.coli* whole-cell model, the massFractionSummary.png, histogramDoublingTime.png, and replication.png output files are of particular interest as these provide the information extracted and graphed with the *M.genitalium* whole-cell model (Figure 4). I also extract a metadata file (‘short-name’) as it records the RNA id of

the gene knocked out in the simulation. To create comparison images (Figure 17), I use Wand [158], a python binding of ImageMagick [159], to make the .png files backgrounds transparent and overlay the images on top of each other (github.com/squishybinary/Ecoli_whole-cell_model_analysis). The model outputs and comparison graphs are assessed manually.

2.2.6 Modelling Scripts

There are six scripts used to run the *M.genitalium* whole-cell model. Three are the experimental files created with each new experiment (the bash script, gene list, experiment list), and three are stored within the whole-cell model and are updated only upon improvement (MGGrunner.m, runGraphs.m, and compareGraphs.m). The bash script is a list of commands for the supercomputer(s) to carry out. Each bash script determines how many simulations to run, where to store the output, and where to store the results of the analysis. The gene list is a text file containing rows of gene codes (in the format 'MG_XXX',). Each row corresponds to a single simulation and determines which genes that simulation should knockout. The experiment list is a text file containing rows of simulation names. Each row corresponds to a single simulation and determines the final location of the simulation output and analysis results.

There is only one script used to run the *E.coli* whole-cell model. The bash script is created with each new experiment and contains a list of commands for the supercomputer to carry out. These commands prepare the supercomputer to execute two main tasks: load the described simulation onto the external database as a series of tasks, and then execute those tasks on the supercomputer until the simulations are complete. The genes to delete are defined in the simulation description in the bash script. The analysis is automatically conducted, and the output is stored automatically by the mode (Section 6.2.4).

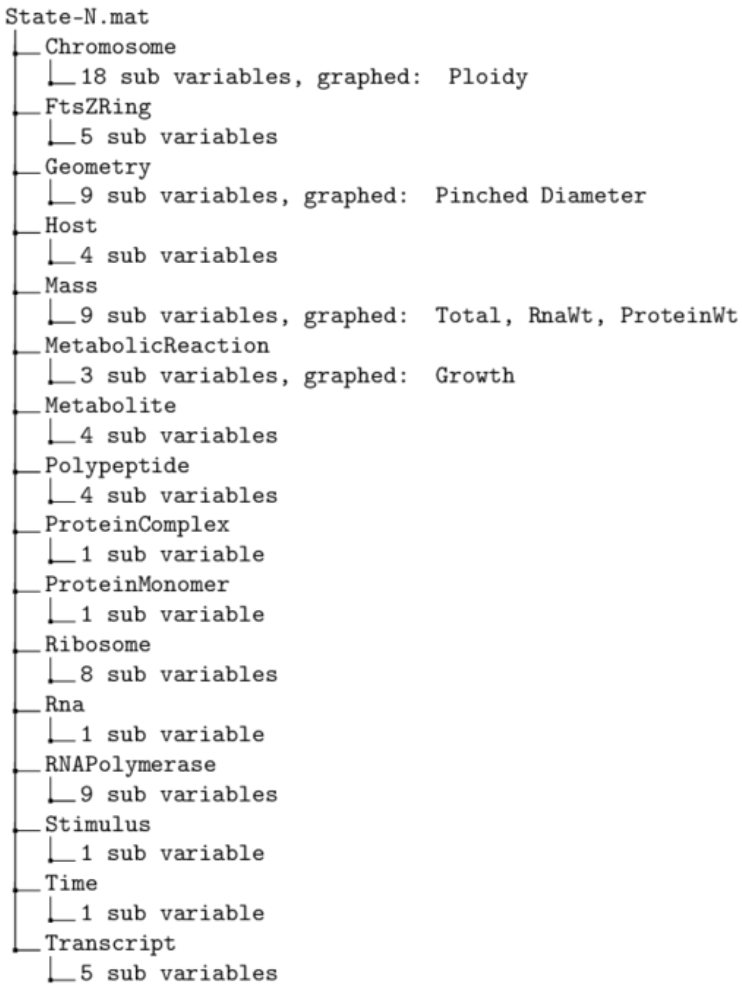

Figure 2. Anatomy of a state file (*M.genitalium* whole-cell model). The data within a *state-XXX.mat* file is organised into 16 cellular variables. The variables graphed in my analysis are: Ploidy as DNA replication, Pinched Diameter as Cell Division, Total as Mass, RnaWt as RNA production, ProteinWt as Protein Production, Growth as Growth.

2.3 Running the *M.genitalium* whole-cell Model

This section (and Section 2.4.5) were written by myself as tutorials for training other members of our research group and for potential future users, as the *M.genitalium* whole-cell model documentation was lacking important information and is now no longer available.

2.3.1 Installing the *M.genitalium* whole-cell Model on a Supercomputer

1. Download the whole-cell model files from github.com/CovertLab/WholeCell
2. Download analysis files from github.com/squishybinary/Analysis_Code_for_Mycoplasma_genitalium_whole-cell_model
3. Create the following directory structure on your institute's supercomputer:
    - /home/USER folder
        - WholeCell-master
            - WholeCell-master
                - -[WholeCell model files]
                - -[runGraphs.m]
                - -[compareGraphs.m]
                - -[WildTypeBackground.fig]
                - src
                    - +edu/+stanford/+covert/+cell/+sim
                        - +runners
                            - -[MGGrunner.m]
        - BlueGem (or NAME OF SUPERCOMPUTER)
            - BlueGemScripts -[*.sh]
            - KOLists -[*_ko.list]
            - ExpLists -[*_exp.list]
    - /projects folder
        - GROUP folder
            - USER folder
                - Output folder
                    - PROJECTfolder
                        - Name of *.sh file folder
                            - 'simname' (see lines of *_exp.list) folder
                                - 1 -[simulation files]
                                - n (depending on number of simulations, see *_exp.list)
                            - WildType (stage dependent)
                                - n
                            - Mutant (stage dependent)
                                - n
                            - pdfs
                            - figs
4. Upload the whole-cell model files to location indicated
5. Upload runGraphs, compareGraphs, WildTypeBackground.fig, MGGrunner
6. Upload generated simulation files (*.sh, *_ko.list, *_exp.list) to location indicated

2.3.2 Running a wild type *M.genitalium* whole-cell Model Simulation

In brief, to manually run the whole-cell model: a new bash script, gene list, and experiment list are created on the desktop computer to answer an experimental question. The supercomputer is accessed on the desktop via ftp software, where the new experimental files are uploaded, the planned output folders are created, and MGGRunner.m, runGraphs.m, compareGraphs.m files are confirmed to be present. The supercomputer is then accessed on the desktop via ssh software, where the new bash script is made executable and added to the supercomputer's queuing system to be executed. Once the experiment is complete, the supercomputer is accessed on the desktop via ssh software, where the results of the analysis are moved to /pdf and /fig folders. These folders are accessed on the desktop via ftp software, where the results of the analysis are downloaded. More detailed instructions are contained within the template bash scripts.

Detailed steps:

1. Download the wild type simulation file from: github.com/squishybinary/Supercomputer_Simulation_Files_for_Mycoplasma_genitalium_whole-cell_model/blob/master/WildTypeSimulation.sh
2. The file is set to run 10 wild type simulations. If you would like to change this:
    a. Modify Line 54 (#SBATCH --array=1-10)
        i. E.g. for one simulation: #SBATCH --array=1
        ii. E.g for seven simulations: #SBATCH --array=1-7
3. In the .sh file, change the "USER" name and the "PROJECTfolder" name to your preferred project name
4. Upload the simulation file (*.sh) to the supercomputer (see directory structure) using ftp software
    a. /BlueGem (or NAME OF SUPERCOMPUTER)
        i. BlueGemScripts -[*.sh]
5. Create a project folder matching the name you gave it Step 3

6. Inside your project folder create the following folders in the displayed structure:
    a. WildTypeSimulation
        i. 1
        ii. 2
        iii. … (create folders equal to the number of simulations running)
        iv. pdfs
        v. figs
7. Connect to the supercomputer using a terminal / shell / command line tool
8. Run the simulation files on the supercomputer:
    a. Make the script executable:

```
chmod u+x WildTypeSimulation.sh
```

    b. Run the script (if your supercomputer uses Slurm)

```
sbatch WildTypeSimulation.sh
```

    c. Check it is running (if your supercomputer uses Slurm)

```
squeue -u [your_username]
```

9. Once the simulations have completed, the resulting files will appear in the respective WildTypeSimulation/N folders.
10. Copy, paste, and run lines 118 and 119 from WildTypeSimulation.sh separately into the terminal / shell / command line to collect all analysis files into the /pdfs and /figs folders, changing the username and projectfolder name

```
find /projects/flex1/USER/output/PROJECTfolder/WildTypeSimulation -type f -iname "*.pdf" -exec mv -t /projects/flex1/USER/output/PROJECTfolder/WildTypeSimulation/pdfs {} +
find /projects/flex1/USER/output/PROJECTfolder/WildTypeSimulation -type f -iname "*.fig" -exec mv -t /projects/flex1/USER/output/PROJECTfolder/WildTypeSimulation/figs {} +
```

11. Download the analysis files from the /pdfs and /figs folders using ftp software.

2.3.3 Running a gene knockout *M.genitalium* whole-cell Model Simulation

Detailed steps:

1. Download the GeneKnockoutSimulation files (*.sh, *_exp.list, and *_ko.list) from: github.com/squishybinary/Supercomputer_Simulation_Files_for_Mycoplasma_genitalium_whole-cell_model
2. In the .sh file, change the “USER” name and the “PROJECTfolder” name to your preferred project name
3. Upload simulations files (*.sh, *_ko.list, *_exp.list) to the supercomputer (see directory structure above):
    a. /BlueGem (or NAME OF SUPERCOMPUTER)
        i. BlueGemScripts -[*.sh]
        ii. KOLists -[*_ko.list]
        iii. ExpLists -[*_exp.list]
4. Create a project folder matching the name you gave it Step 2
5. Inside your project folder create the following folders in the displayed structure:
    a. GeneKnockoutSimulation
        i. MG_007
            A. 1
            B. 2
            ..
            C. 10
        ii. WildType
            A. 11
        iii. Mutant
            A. 12
        iv. pdfs
        v. figs
6. Connect to the supercomputer using a terminal / shell / command line tool
7. Run the simulation files on the supercomputer:
    a. Make the script executable:

```
chmod u+x GeneKnockoutSimulation.sh
```

    b. Run the script (if your supercomputer uses Slurm)

```
sbatch GeneKnockoutSimulation.sh
```

    c. Check it is running (if your supercomputer uses Slurm)

```
squeue -u [your_username]
```

8. Once the simulations have completed, the resulting files will appear in the respective numbered folders.
9. Copy, paste, and run lines 118 and 119 from GeneKnockoutSimulation.sh separately into the terminal / shell / command line to collect all analysis files into the /pdfs and /figs folders, changing the username and projectfolder name

```
find /projects/flex1/USER/output/PROJECTfolder/GeneKnockoutSimulation -type f -iname "*.pdf" -exec mv -t /projects/flex1/USER/output/PROJECTfolder/GeneKnockoutSimulation/pdfs {} +
find /projects/flex1/USER/output/PROJECTfolder/GeneKnockoutSimulation -type f -iname "*.fig" -exec mv -t /projects/flex1/USER/output/PROJECTfolder/GeneKnockoutSimulation/figs {} +
```

10. Download the analysis files from the /pdfs and /figs folders using ftp software.

2.4 Chapter 4 and 5 Specific Methods

2.4.1 Code Availability

The code for the Minesweeper genome design algorithm, scripts for statistical analysis, scripts for analysing GO terms, my custom simulation runner, analysis scripts, a template bash script, as well as bash scripts and text files used to generate the simulations are available on Github (github.com/GriersonMarucciLab).

2.4.2 Statistics

I used the R binom package (rdocumentation.org/packages/binom) to conduct one-tailed binomial proportion confidence intervals on 41 genes showing inconsistent results (success ranging from 6 to 9 replicates, out of a total of 10 replicates). I used binom.confit.exact (Pearson-Klopper) using 95% CIs, producing for: 6/10 replicates [0.26, 0.87], 7/10 replicates [0.34, 0.93], 8/10 replicates [0.44, 0.97], 9/10 replicates[0.55, 0.99]). I graphed these results in R and in Python using Seaborn (seaborn.pydata.org/) , the exact values, code, and graphs produced are available online (Supplementary Data 2 [56]).

2.4.3 Data Analysis Process

For *in-silico* experiments conducted using Minesweeper, simulations were automatically classified solely by division, which can be analysed from cell width or the endtime of the simulation. Further analysis, including: cross-comparison of single gene knockout simulations, comparison to Karr *et al.*'s [6] results, analysis of Minesweeper and GAMA genomes (genetic content and similarity, behavioural analysis, phenotypic penetrance, gene ontology), and identification and investigation of high and low essentiality genes and groupings, were completed manually.

The GO biological process terms were downloaded from Uniprot [160] (strain ATCC 33530/NCTC 10195), processed by a created script (github.com/squishybinary/Gene_Ontology_Comparison_for_Mycoplasma_genitalium_whole-cell_model), then organised manually into tables of GO terms that were unaffected, reduced, or removed entirely by gene deletions.

2.4.4 Data Availability

The databases used to design the *in-silico* experiments, and compare the results to, includes Karr *et al.* [6] and Glass *et al.* [61] Supplementary Tables, and Fraser *et al.* *M.genitalium* G37 genome [58] interpreted by KEGG [132] and UniProt [160] as strain ATCC 33530/NCTC 10195. The output .fig files for all simulations referenced in Chapter 4 are available from the group's Research Data Repository (data-bris) at the University of Bristol, with the identifier doi.org/10.5523/bris.1jj0fszzrx9qf2ldcz654qp454 [161]. All of the Minesweeper genome simulations require 4.2 TB of data. These are stored by the group to be available for request.

2.4.5 Using the Minesweeper Algorithm with the *M.genitalium* whole-cell model

Minesweeper conducts whole-cell model simulations in three step cycles: design (algorithms select possible gene deletions); simulate (the genome minus those deletions); and test (analyse resulting cell). Simulations that removed the most genes from the *in-silico* genome and produced dividing model cells proceed to the next cycle, increasing the number of gene deletions and producing progressively smaller genomes. Minesweeper is a four-stage algorithm inspired by divide and conquer algorithms [162], stages one to three are sequential, with stage four repeating until Minesweeper stops. It

initially deletes genes in groups but eventually deletes individual genes, and only deletes non-essential genes (determined by single gene knockout simulations). It uses between 8 and 359 CPUs depending on the stage, with data storage handled by user-submitted information and simulation execution conducted manually. It is written in Python3 and consists of four scripts (one for each stage). It is run manually on a non-dedicated desktop, with its output determining which simulations to be run on the supercomputer. It uses no external libraries, so should be able to be run on any modern operating system (as they come with Python preinstalled) via a terminal. Each stage/script requires a text file(s) as input, with each stage outputting simulation files. These are run on a supercomputer and the automatically produced summary file is used as input for the next stage on the desktop, with progress recorded in the deletion log in /OUTPUT_final. See Chapter 4 for a more in-depth explanation of the Minesweeper algorithm.

2.4.5.1 Minesweeper Demo

This demo was produced as part of the publication process for Minesweeper. It explains a step by step process for completing a full cycle of the Minesweeper algorithm using the *M.genitalium* whole-cell model. For code and equipment required see Section 2.2.2 Model Availability and Section 2.2.3 Equipment.

The test data provided with the demo was produced by completing stages one to four of Minesweeper using the averaged single gene essentiality data from Table 6 (Section 4.4.1), producing 2310 *in-silico* minimal genomes with stage four repeating three times.

2.4.5.2 Steps to Install Minesweeper

1. Download / unzip Minesweeper code and test data
    a. github.com/GriersonMarucciLab/Minesweeper
2. Make a copy of the Minesweeper_0.9 folder, rename it appropriate to your project, and place it in your preferred location on your local desktop.
3. Minesweeper requires Python3. To check you have Python3 installed on your local desktop, type into your terminal / IDE (integrated development environment):

```
python --version
```

4. If you need to install python3, visit python.org/downloads/ for the appropriate installer.

2.4.5.3 First Stage

The first stage of Minesweeper conducts individual protein-coding gene knockouts, removing complete/high essential genes as deletion candidates. The first stage of Minesweeper is optional, if you already have single gene knockout simulation results you can proceed to the second stage.

Detailed Steps:

1. Check the first required text file (genes.txt, containing the protein-coding gene codes in the model) is already present in a preexisting folder structure (INPUT_script_1).
2. Skip this step if this is your first time running the algorithm or you want to run using the default settings:
    a. If you would like to create a custom file of genes to attempt to knockout from the *M.genitalium in-silico* genome:
        i. Format (one per line): MG_XXX
        ii. Save as: genes.txt
        iii. Save to: INPUT_script_1 folder

b. If there are genes you wish to be excluded from deletion (exclusion starts from the second stage):
    i. Format (one per line): MG_XXX
    ii. Save as: exclusionlist.txt
    iii. Save to: INPUT_script_2 folder
c. If this is your second time running the algorithm and you already have single gene knockout results for all genes (or you have an average phenotype result from multiple simulations for each single gene):
    i. Create a custom output file for the first stage (see templatescript folder for a template)
        1. Format: sim_number [tab] time [tab] outcome
            a. Time values: 13.89 for no division
            b. Time values: 11.01(or another random value less than 13.89) for divided
            c. Outcome values: nodivision OR divided
            d. Save as: inputko1_endtimes.txt
            e. Save to: INPUT_script_2 folder
    ii. You also need to create a false experiment list for the first stage (see templatescript folder for a template):
        1. Format:
            a. lines containing 'inputko'\n * number of genes
            b. and two lines at the end: 'wild type'\n, 'mutant'
            c. Save as: mineinputko1_exp.list
            d. Save to: OUTPUT_script_1 folder

3. In either a terminal or an IDE:
    a. Change the directory to your project folder in your preferred location
    b. Run the code by typing:

```
python script1.py
```

4. The code will output generated simulation files and save actions taken to the log file.
    a. See generated simulation files: *.sh , *_exp.list, *_ko.list in OUTPUT_script_1 folder
    b. Log file: deletionlog.txt in OUTPUT_final
5. Upload generated simulation files (*.sh, *_ko.list, *_exp.list) to the supercomputer (see directory structure above):
    a. /BlueGem (or NAME OF SUPERCOMPUTER)
        i. BlueGemScripts -[*.sh]
        ii. KOLists -[*_ko.list]
        iii. ExpLists -[*_exp.list]
6. Run the simulation files on the supercomputer (if it uses the Slurm queuing system see lines 10 - 35 of *.sh generated simulation file)
7. Once the simulations have completed, the resulting summary files will appear in the name-of-.sh folder (see directory structure above). Download the file and place it in the correct folder:
    a. inputkoN_endtimes.txt
    b. inputkoN_endtimes.txt in /INPUT_script_2 folder

2.4.5.4 Second Stage

The second stage sorts the singly non-essential genes into 26 deletion segments (100%, 90%A, 90%B, 80%A, 80%B, 70%A, 70%B, 60%A, 60%B, 50%A, 50%B, 33%A-C, 25%A-D, 12.5%A-H). The A segments start from the top of the list of genes, whereas the B segments start from the bottom of the gene list. Deletion segments that can be removed and still produce a dividing cell are carried forward.

Detailed steps:

1. In either a terminal or an IDE:
    a. Change the directory to your project folder in your preferred location
    b. Run the code by typing:

```
python script2.py
```

2. The code will output generated simulation files and save actions taken to the log file.

    a. See generated simulation files: *.sh , *_exp.list, *_ko.list in OUTPUT_script_1 folder
    b. Log file: deletionlog.txt in OUTPUT_final
3. Upload generated simulation files (*.sh, *_ko.list, *_exp.list) to the supercomputer (see directory structure above):
    a. /BlueGem (or NAME OF SUPERCOMPUTER)
        i. BlueGemScripts -[*.sh]
        ii. KOLists -[*_ko.list]
        iii. ExpLists -[*_exp.list]
4. Run the simulation files on the supercomputer (if it uses the Slurm queuing system see lines 10 - 35 of *.sh generated simulation file)
5. Once the simulations have completed, the resulting summary files will appear in the name-of-.sh folder (see directory structure above). Download the file and place it in the correct folder:
    a. divideko_endtimes.txt
    b. divideko_endtimes.txt in /INPUT_script_3

2.4.5.5 Third Stage

The third stage progresses with the three largest deletion segments that produced a dividing cell, these three variants are referred to as red, yellow, blue. These perform as replicates and as a check on if the results are converging. The three variants are matched with smaller, dividing, non-overlapping segments using a list of allowed matches (implementation is detailed in Section 4.3.7). A powerset is generated (i.e. a set containing all possible unique combinations of the matched deletion segments, including zero and individual deletion segments) and each of the deletion combinations is removed from an individual *in-silico* cell and simulated.

Detailed steps:

1. In either a terminal or an IDE:
    a. Change the directory to your project folder in your preferred location
    b. Run the code by typing:

```
python script3.py
```

2. The code will output generated simulation files and save actions to the log file.
    a. See generated simulation files: *.sh , *_exp.list, *_ko.list in OUTPUT_script_1 folder
    b. Log file: deletionlog.txt in OUTPUT_final
3. Upload generated simulation files (*.sh, *_ko.list, *_exp.list) to the supercomputer (see directory structure above):
    a. /BlueGem (or NAME OF SUPERCOMPUTER)
        i. BlueGemScripts -[*.sh]
        ii. KOLists -[*_ko.list]
        iii. ExpLists -[*_exp.list]
4. Run the simulation files on the supercomputer (if it uses the Slurm queuing system see lines 10 - 35 of *.sh generated simulation file)
5. Once the simulations have completed, the resulting summary files will appear in the name-of-.sh folder (see directory structure above). Download the file and place it in the correct folder:
    a. conquerko_COLOUR_0_endtimes
    b. conquerko_COLOUR_0_endtimes in /INPUT_script_4X

2.4.5.6 Fourth Stage

The fourth stage is cyclical. The largest deletion combination from the third stage generates a remaining gene list (those yet to be deleted). The remaining genes are split into eight groups and a powerset is generated. Each deletion combination from the powerset is individually appended to the current largest deletion combination and simulated. The simulation results update the largest deletion combination, which is used to generate a new remaining gene list, starting the next cycle. If none of the deletion combinations produce a dividing cell, the remaining genes are singly appended to the largest deletion combination, removed and simulated. The individual remaining genes that do not produce a dividing cell are excluded for a cycle and a reduced remaining gene list is generated, which is used for the next cycle.

The fourth stage continues until there are eight or fewer remaining genes (where a final appended powerset is run) or all individually appended remaining genes do not produce a dividing cell. Both outcomes result in a list of deleted genes and identified low essential genes.

Repeat these steps until Minesweeper finishes:

1. In either a terminal or an IDE:
    a. Change the directory to your project folder in your preferred location
    b. Run the code by typing:

```
python script4X.py
```

2. The code will output generated simulation files and save actions taken to the log file. Check the log file for progress and whether another round of simulations is required.
    a. See generated simulation files: *.sh , *_exp.list, *_ko.list in OUTPUT_script_1 folder
    b. Log file: deletionlog.txt in OUTPUT_final
3. Upload generated simulation files (*.sh, *_ko.list, *_exp.list) to the supercomputer (see directory structure above):
    a. /BlueGem (or NAME OF SUPERCOMPUTER)
        i. BlueGemScripts -[*.sh]
        ii. KOLists -[*_ko.list]

iii. ExpLists -[*_exp.list]

4. Run the simulation files on the supercomputer (if it uses the Slurm queuing system see lines 10 - 35 of *.sh generated simulation file)
5. Once the simulations have completed, the resulting summary files will appear in the name-of-.sh folder (see directory structure above). Download the file and place it in the correct folder:
    a. Potential file names:
        i. gapko_COLOUR_1_endtimes
        ii. gapko_COLOUR_2_endtimes
        iii. gapko_COLOUR_N_endtimes
    b. gapko_COLOUR_N_endtimes in /INPUT_script_4X
6. Repeat Steps 1 to 5, though pay attention to the log file (Step 2) which will tell you when Minesweeper has finished.

The reason for selecting eight groups and three variants is that a powerset of eight produces 256 unique combinations. Three variants each with 256 simulations (768 total) represents 85% of the capacity of BlueGem. A set of nine groups with three variants (1536 simulations total) is 170% the capacity of BlueGem. Queueing systems mean that you do not require this total number of CPUs to be available, but the execution time is multiplied as you wait for the simulations to process. The number of variants and groups can be lowered or increased depending on the number of CPUs available. To do so, change the calculation and list generation values in the eightPanelGroupingsGeneration function in the fourth script.

2.5 Chapter 6 Specific Methods

2.5.1 Installing the *E.coli* whole-cell Model on a Supercomputer

I have made a detailed list of step by step instructions methods for installing the *E.coli* whole-cell model onto a PBS based supercomputer (e.g. BlueCrystal) available on Github (github.com/squishybinary/Ecoli_whole-cell_model_analysis). Special thanks are due to Dee (Ms Dianaimh Greene) of the Advanced Computing Research Centre, who created the original version of the step by step instructions.

In brief, the main steps are:

1. Download the whole-cell model files from github.com/CovertLab/WholeCellEcoliRelease
2. Upload the model to the supercomputer via ftp
3. Install python tools
4. Modify the startup process to load specific tools from the supercomputer
5. Create a "virtual environment", a separate, enclosed space for installing tools, around the *E.coli* whole-cell model

Within the virtual environment:

6. Install python
7. Install a common linear algebra operations tool
8. Install required libraries and tools
9. Test installations
10. Change the matplotlib backend to enable correct graphing by the model
11. Compile the model's code
12. Run tests to see if the model is functioning correctly

2.5.2 Running the *E.coli* whole-cell Model on a Supercomputer

1. Upload FireWorks .yaml files using FTP (changing the capitalised values) to:
    a. /HOME/USER/Folder_containing_model/wcEcoli/wholecell/fireworks
2. Create and upload a FireWorksBox bash script (Section 6.2.4) to the supercomputer via FTP (changing the capitalised values):
    a. /HOME/USER/Folder_containing_model/wcEcoli/wholecell/fireworks
3. Login to the supercomputer using ssh
4. Create the following directories in /fireworks
    a. /logs
        i. /launchpad
        ii. /qadapter
5. Make the FireWorksBox executable and run it:

```
cd wcEcoli/wholecell/fireworks
chmod u+x FireWorksBox_1.sh
./FireWorksBox_1.sh
```

Chapter 3 - Developing Analysis For and Running the *M.genitalium* whole-cell Model

3.1 Statement of Collaboration

I received initial training in running the *M.genitalium* whole-cell model on University of Bristol's BlueGem supercomputer from PhD student Oliver Chalkley.

3.2 Aims

The *M.genitalium* whole-cell model is the first model of its kind, published relatively recently, and requires interdisciplinary research teams to implement the model and analyse the resulting data. Due to this, when starting this work in 2016, the number of researchers in the whole-cell modelling community was small, documentation was present but sparse, and standards had not been fully finalised. I took this opportunity to produce an analysis process for our group and to standardise our running of the whole-cell model and the analysis of the model's output.

3.3 Developing Analysis for *M.genitalium* whole-cell Model Simulations

Oliver Chalkley, the first PhD student in the group, had already installed the *M.genitalium* whole-cell model on the University's supercomputers (BlueGem and BlueCrystal) when I joined the group. I focused my efforts on developing an analysis process for the simulation data. No formalised process currently existed within the group, so I started with the analysis conducted by the initial publication [6]. "Figure 6B" in Karr *et al.*'s paper (Figure 3) shows the potential range of *in-silico* cell phenotypes. Single gene knockout simulations were conducted and classified according to behaviour in five criteria (growth, protein mass, RNA mass, DNA mass, septum size) in comparison to wild type (non gene edited) simulations, over the length of the simulation. Each column in Figure 3 corresponds to a different phenotype class and shows a representative gene knockout, with the criteria for that classification highlighted in orange. Two categories, quasi-essential and other, were indicative of failures over multiple generations, which were simulated using only the growth sub-model; the *M.genitalium* whole-cell model as a whole is not capable of multiple generation simulations.

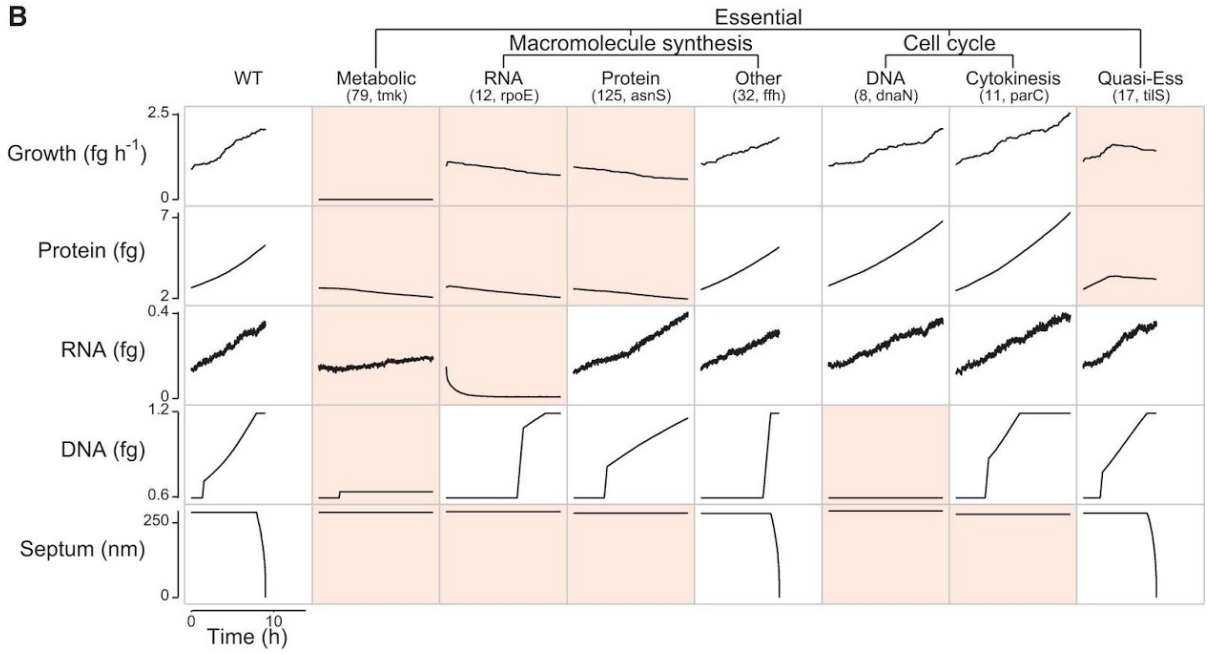

Figure 3. "Figure 6B". Figure and legend replicated from Karr *et al.* 2012 [6]. Single gene disruption strains were grouped into phenotypes (columns) according to their capacity to grow, synthesize protein, RNA, and DNA, and divide (indicated by septum length). Each column depicts a representative *in-silico* cell for each phenotype. Dynamics significantly different from wild type are highlighted in orange. The identity of the representative cell and the number of disruption strains in each category are indicated in parentheses.

The code used to create "Figure 6B" is 3392 lines long (SingleGeneDeletions.m) [163]. To start creating an analysis process, I extracted 22 functions from this file, and other files within the model, essential to loading, processing, classifying, and graphing the simulation data. This proved difficult due to the lack of accompanying documentation. For example, the plotOverview function from SingleGeneDeletions.m did correspond to "Figure6B". But, as can be seen in the code below, there is a lack of explanatory notes or commentary in the code and several categories (underlined) are not present in the published literature. Substantial work was required to understand the dependencies and architecture in the model data that the plotting functions drew upon.

```
catLabels = {
        'WT'            SingleGeneDeletions.WILD_TYPE
        'Energy'        SingleGeneDeletions.DECOMPOSING
        'Metabolic'     SingleGeneDeletions.NON_GROWING
        'RNA'
SingleGeneDeletions.DECAYING_GROWTH_NON_RNA_PROTEIN
        'Protein'       SingleGeneDeletions.DECAYING_GROWTH_NON_PROTEIN
        'Other'         SingleGeneDeletions.NON_PERPETUATING
        'DNA'           SingleGeneDeletions.NON_REPLICATIVE
        'Cytokinesis'   SingleGeneDeletions.NON_FISSIVE
        'Term Org'      SingleGeneDeletions.NO_TERMINAL_ORGANELLE
        'Damaged'       SingleGeneDeletions.TOXIN_ACCUMULATION
        'Quasi-Ess'     SingleGeneDeletions.SLOW_GROWING
        'Non-Ess'       SingleGeneDeletions.NON_ESSENTIAL
        };
propLabels = {
        {'NTP'}             'ntps'
        {'Growth (fg h^{-1})'}  'growth'
        {'Protein (fg)'}        'proteinWt'
        {'RNA (fg)'}            'rnaWt'
        {'DNA (fg)'}            'dnaWt'
        {'Septum (nm)'}         'pinchedDiameter'
        {'Term Org (fg)'}       'terminalOrganelleWt'
        {'Damaged' 'Prot'}      'damagedProteins'
        };
```

This first version of my analysis process consisted of 3 files: runWholeCellAnalysis, processRawData, and MultiGeneOneExp (github.com/squishybinary/Original_whole-cell_analysis), calling upon the 22 extracted functions, some of which were modified to function independently.

Two problems arose. Firstly, it became evident that the files that SingleGeneDeletions.m were manipulating were different to those produced by processRawData. A processing step was missing. This was later confirmed by gaining access to the original publication's raw data (archive.simtk.org/WholeCell/simulation/output/runSimulation/), where it appeared multiple simulations of each gene was packaged together and averaged (the function that completed this processing step remained elusive).

Secondly, Karr *et al.*'s code required a strict naming convention, based on single gene code names (e.g. MG_006.mat), implementing deletions, conducting analysis, and saving files based on the specific filename. This process was not adaptable to multi-gene knockouts.

This led me to develop my own analysis process from scratch, recreating the classification by pulling the required data directly from the simulation output and creating new graphs that matched "Figure 6B". This analysis process consisted of three scripts: runGraphs, compareGraphs, and WildTypeBackgroundFig (github.com/squishybinary/Analysis_Code_for_Mycoplasma_genitalium_whole-cell_model).

runGraphs.m carries out the initial analysis of a *M.genitalium* whole-cell simulation, replicating "Figure 6B". It plots growth, protein weight, RNA weight, DNA replication, cell division, and records several experimental details (Figure 4). Outputs are saved as MATLAB .fig files and .pdfs.

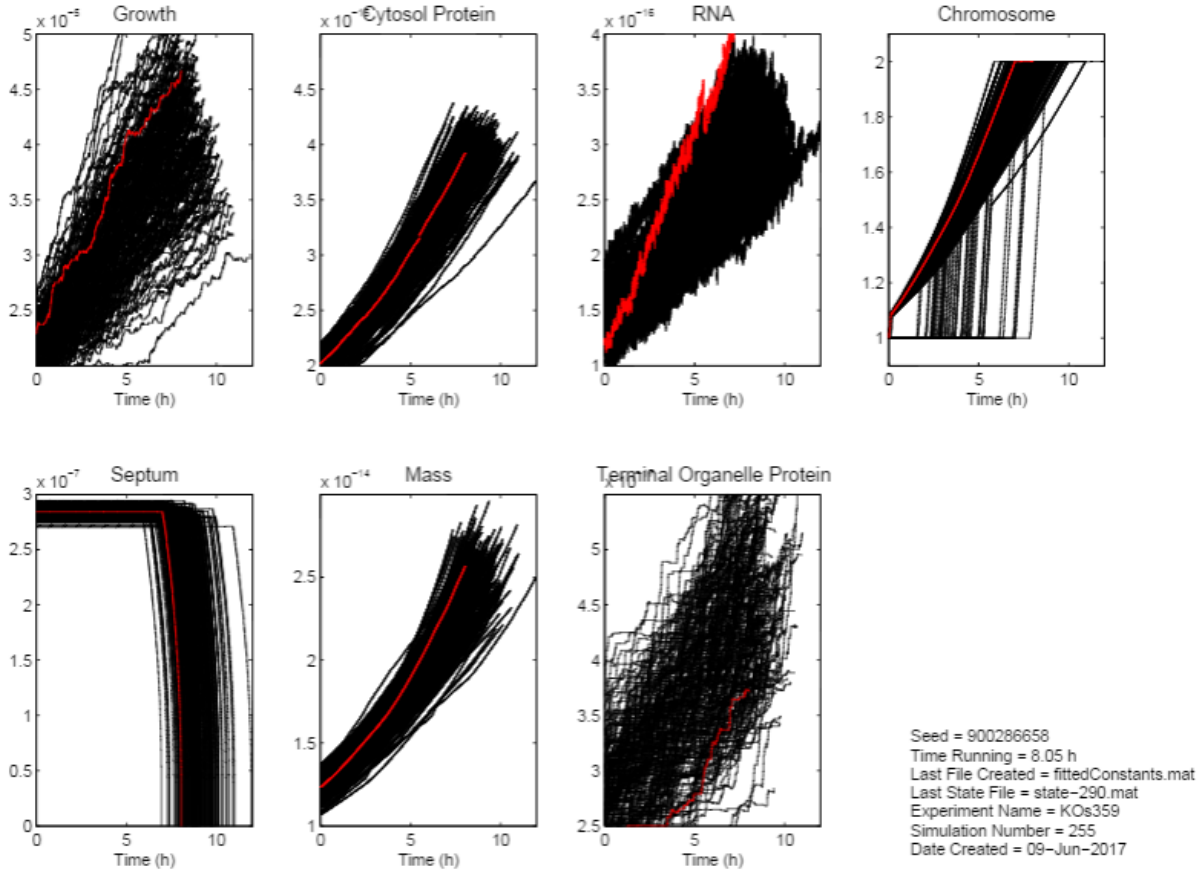

Figure 4. Initial analysis of a single gene knockout. This particular gene is non-essential, as the simulation (in red) performs as expected for a wild type cell (200 wild type simulations in black). Simulation specific data extracted includes seed number, running time, last file created, experiment name, simulation number and date completed.

compareGraphs.m overlays the output of the *M.genitalium* whole-cell simulation on the WildTypeBackground.fig, the collated outputs of 200 wild type *M.genitalium* simulations. Outputs are saved as MATLAB .fig files and .pdfs.

WildTypeBackgroundFig.m produces the WildTypeBackground.fig file by overlaying the outputs of 200 wild type *M.genitalium* simulations.

This analysis process has received minor updates throughout my PhD but is otherwise still used for analysing output from the *M.genitalium* whole-cell model.

3.4 Running the *M.genitalium* whole-cell Model

The *M.genitalium* whole-cell model is written in Matlab. The line of code that actually conducts the simulation is:

```
matlab $options -r "diary('${OutDir}/${Experiment}/${Sim}/diary.out');
addpath('${Master}'); setWarnings(); setPath(); runSimulation('runner','MGGRunner',
'logToDisk',true, 'outDir', ${OutDir}/${Experiment}/${Sim}', 'seedIncrement',
'${SeedInc}'); diary off; exit;"
```

This can be translated as:

- Monitor and record logs of the simulation as it is running

```
diary('${OutDir}/${Experiment}/${Sim}/diary.out');
```

- Let the supercomputer know where the model files are

```
addpath('${Master}');
```

- Turn off Matlab warnings and let Matlab know where the model files are

```
setWarnings();setPath();
```

- Start the simulation (with the following variables):

runSimulation(

  - Use my custom runner file to increment the initial conditions correctly

'runner','MGGRunner',

  - Save the output of the simulation

'logToDisk',true,

  - To this location

'outDir','${OutDir}/${Experiment}/${Sim}',

  - Increment the initial conditions

'seedIncrement','${SeedInc}')

  - Finish logging once runSimulation is complete

;diary off;exit;"

Running simulations on the supercomputer requires creating three files: a bash script, an experiment name list, and a gene knockout list. Bash scripts contain the simulation command (outlined above) and Slurm commands (in the format '${..}'). Slurm is the queuing system used on BlueGem and the commands are interpreted to control the execution, and handle the output of the model (Figure 5). The experiment name list and the gene knockout list contain the information required to run gene knockout simulations, which are extracted by the bash script using the number of the simulation as a line number.

```
# A Supercomputer value that assigns each simulation a number, that doubles as a line number for the list files
Sim=${Slurm_ARRAY_TASK_ID}

# Gene Knockout Variables
# Experiment name list location
experiment_list=/home/USER/BlueGem/ExpLists/GeneKnockoutSimulation_exp.list

# Extract names from experiment list
Experiment=$(awk NR==${Slurm_ARRAY_TASK_ID} ${experiment_list})
```

```
# Gene knockout list location
ko_list=/home/USER/BlueGem/KOLists/GeneKnockoutSimulation_ko.list

# Fetch the Gene knockouts from the knockout list
Gene=$(awk NR==${Slurm_ARRAY_TASK_ID} ${ko_list})
```

An additional variable is added to runSimulation command to conduct gene knockouts:

```
'koList',{{${Gene}}});
```

Although the code says 'Gene' it can refer to one gene or hundreds of genes to be knocked out. Running the model on a supercomputer in this way allows for simulating large numbers of *in-silico* cells in parallel.

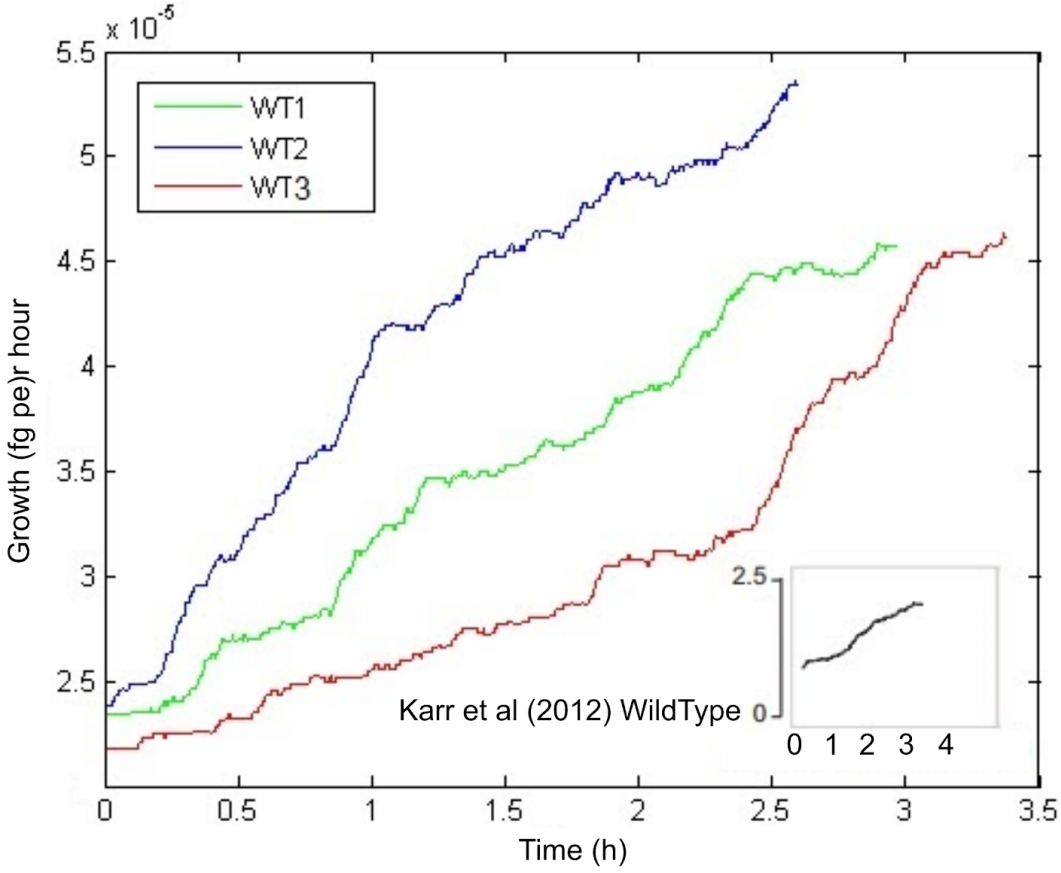

Figure 5. My first *M.genitalium* whole-cell model simulations. These three wild type simulations were run on BlueGem and graphed manually in Matlab, with the previously reported *in-silico* wild type growth pattern [6] shown in-picture.

These three scripts (the bash script and the two lists) have to be created anew for each new experimental simulation series. Combined with the three analysis scripts, these six scripts produce simulations on the supercomputer and analysis plots of the resulting simulated cells (Figure 4). More detailed information is provided in Section 2.3.2 and Section 2.3.3.

Now that I could produce comparable analysis plots, I generated a decision tree (Figure 6) to classify gene knockouts consistently into phenotypic categories. There are seven possible phenotypes caused by knocking out genes in *M.genitalium* whole-cell model simulations: i) non-essential, if a dividing cell is produced; ii) slow growing, if a dividing cell has started the process of division but has not completed before the simulation ends; and essential if a non-dividing cell is produced due to iii) a DNA replication mutation, iv) a RNA production mutation, v) a protein production mutation, vi) a metabolic mutation (DNA, RNA, and protein production are all affected), or vii) a division mutation (labelled septum mutation in Figure 6).

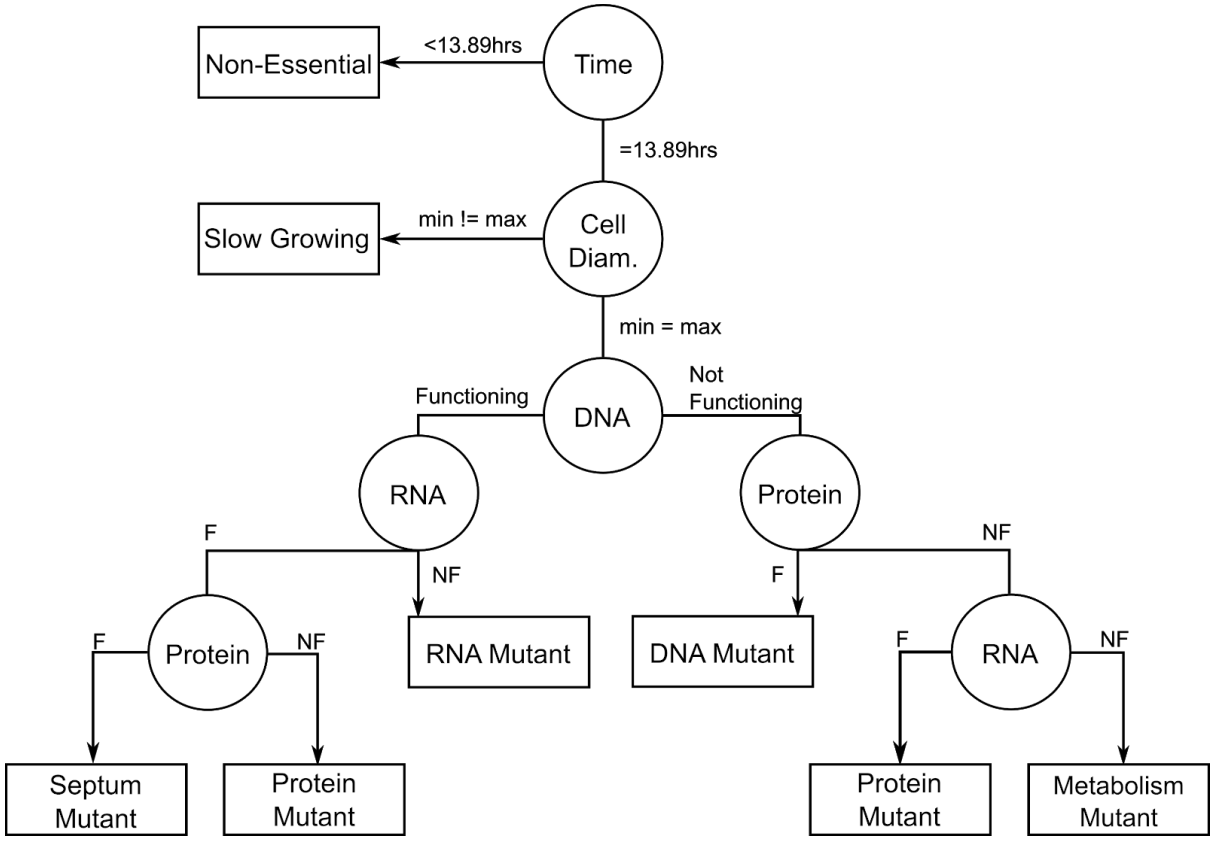

Figure 6. Phenotypic categorisation decision tree for simulations. 13.89 hours is the simulated time allowed before the simulation ends. Cell diameter measures whether the cell started division: yes if min != max, no if min = max. Functioning / F denotes that the cellular function is within the range of behaviour for wild type cells.

3.5 Issues Running the *M.genitalium* whole-cell Model

During initial testing of gene deletion simulations I discovered an error in one of the groups previous scripts (koRunner.m). This error was preventing the simulations from randomly generating initial conditions ("seeding") properly, as the random generation relied on using the current time and date to the hour. BlueGem can spawn upwards of 100 simulations a second. I created MGGRunner.m to artificially increment date time by a number of seconds equal to the number assigned to the simulation by the supercomputer (Figure 7), preventing multiple simulations starting with identical initial conditions.

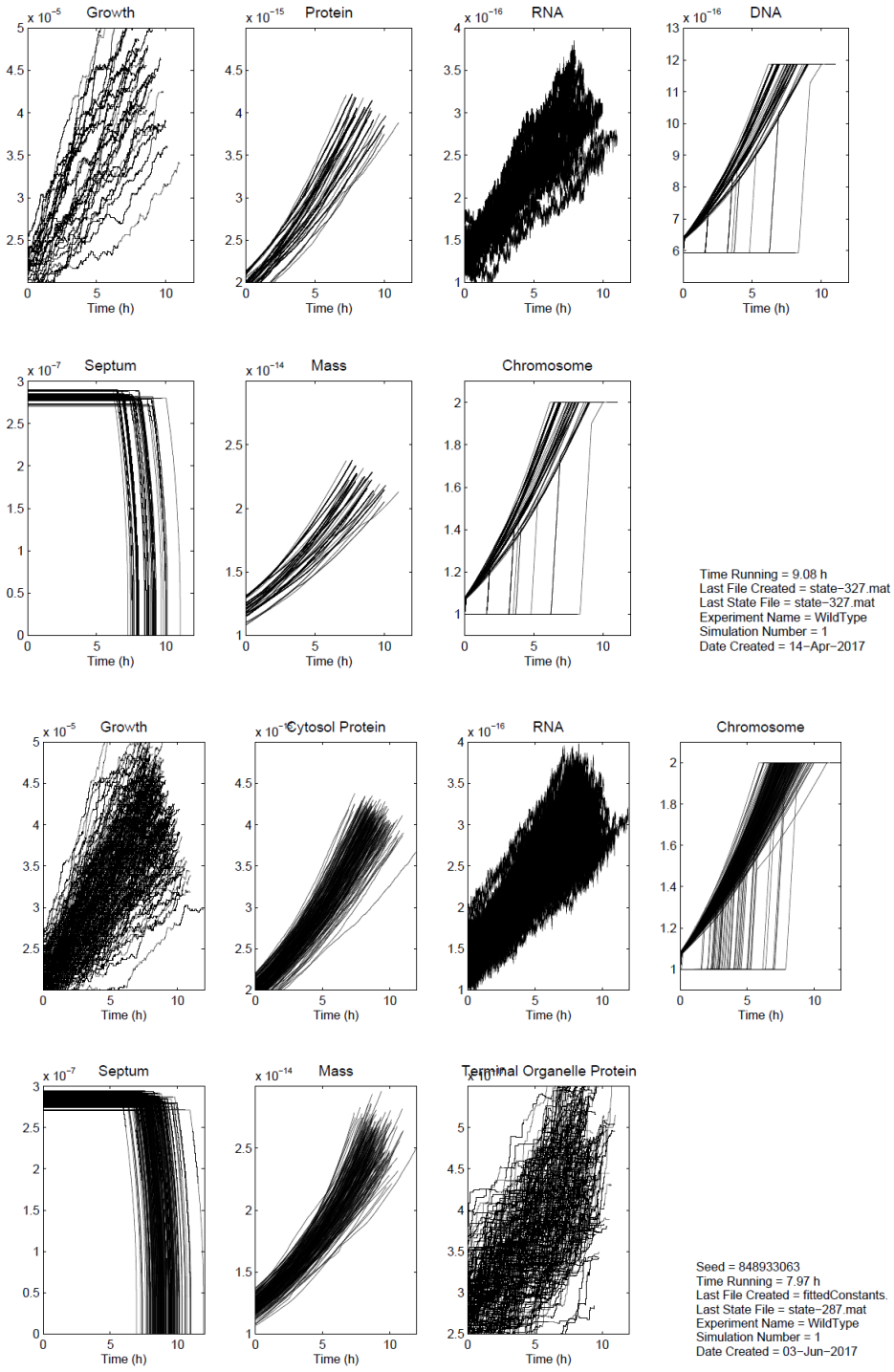

Figure 7. Fixing random “seeding”. (Upper) 200 wild type simulations run using KoRunner.m without working random “seeding”, reducing the number of unique (plottable) simulations. (Lower) 200 wild type simulations with working random “seeding”, using MGGRunner.m. The Chromosome plot had been replaced with Terminal Organelle Protein, as Chromosome displayed identical information to DNA.

An additional investigation identified the reason behind an unexpectedly high incidence of crashes I was observing for *M.genitalium* whole-cell model simulations. Initially thought to be the number of gene deletions causing problems, the gene “MG_469” was found to cause the simulation to crash when knocked out *in-silico* (running on BlueGem using Matlab R2013b). This was confirmed by re-running simulations while excluding “MG_469” from being deleted, resulting in a dramatic reduction in the number of simulations that crashed. It was later discovered that “MG_254” also caused simulations to crash, but only after 5 hours of simulation time. Both “MG_469” and “MG_254” were excluded from future gene deletions. According to Jonathan Karr, the original developers and other users reported gene deletions that caused simulation crashes, with different versions of Matlab potentially being linked to different causal gene deletions. However, the modelling cause for this was not discovered (*personal communication*).

To confirm the status of unmodelled genes (which were not originally modelled due to unknown function) within the *M.genitalium* whole-cell model, new simulations were run. In theory, these genes are listed within the model but do not have any associated functions. The 124 unmodelled genes were knocked out within the model and produced a successfully dividing cell that was indistinguishable from a wild type *in-silico* cell, confirming that the unmodelled genes were not functionally implemented within the model.

3.6 Discussion

Having established a repeatable analysis process that could be used to categorise the phenotypes of the *in-silico* cells reliably, I began developing a genome design algorithm and conducting my first experimental research, attempting to produce *M.genitalium* *in-silico* minimal genomes.

Chapter 4 - Minesweeper: A Minimal Genome Design Algorithm

4.1 Statement of Collaboration

Sections of this chapter are adapted from a co-first authored paper with Oliver Chalkley. We worked in parallel, Oliver was also developing a genome design algorithm, named GAMA (Guess, Add, Mate Algorithm) and producing minimal genome simulations. However, I analysed all the simulation data that was produced and presented here. I was also the lead author, drafting the paper and managing the writing and editing process, with support from other members of the team and advice from my supervisors; the text reproduced below is my own work. Oliver's work, the development of GAMA and the data it produced, is presented in his thesis and is available in published papers [56,164] and will not be discussed here.

- Rees-Garbutt, J. *et al*. Designing minimal genomes using whole-cell models. *Nat. Commun*. **11**, 836 (2020)
    - Lead author
- Rees-Garbutt, J., Grierson, C. & Marucci, L. Testing theoretical minimal genomes using whole-cell models. *bioRxiv.* doi:10.1101/2020.03.26.010363 (2020).
    - Lead author
- Rees-Garbutt, J. Minesweeper Genome Design Algorithm. *Github*. https://github.com/squishybinary/Minesweeper (2020).
    - Lead author

4.2 Aims

As genome design algorithms for genome engineering are rare (Section 1.3.4.b), I chose to start developing my genome design algorithm from the basis of genome segmentation used by current *in-vivo* genome engineering efforts (Section 1.3.2.d) and to focus on the development of minimal genomes. Minimal genomes are a good proof of concept for genome engineering as the success criteria is easy to assess (did the cell divide or not) and it does not require a large amount of genetic knowledge (which we currently lack) to make progress. The minimal genome design algorithm I created

produced 1000s of *in-silico* cells on the University of Bristol's BlueGem supercomputer, resulting in the creation of an *M.genitalium in-silico* minimal genome which, if biologically correct, predicts an *in-vivo* genome smaller than *JCVI-Syn3.0*; a bacterium with, currently, the smallest genome that can be grown in pure culture [60].

4.3 Algorithm Development

4.3.1 Rationale

My initial thoughts around producing a genome design algorithm were: I wanted to produce results quickly and I wanted to develop an algorithm based loosely on the practicalities and constraints of recent genome engineering work [12] that breaks genomes down into workable sizes for laboratory manipulations (Section 1.3.2.d). The divide and conquer methodology [165] I settled on deals with both of these requirements. It involves dividing the whole problem into smaller sub-problems, solving those sub-problems, and combining the solutions to solve the whole problem; very similar in concept to breaking down genomes in a laboratory. As you can vary the size of the sub-problems, it also allows you to investigate a number of large-scale deletions in the first stages of the algorithm which produces results quickly. As an additional design consideration, I decided to focus on genes that were identified as non-essential by *in-silico* single knockout (i.e. their removal did not prevent cell division) to make as much progress as possible in the earlier stages, by not producing a large percentage of *in-silico* cells that failed to divide.

As the model only conducted single generation simulations, the algorithm would have to make progress using a design (select possible gene deletions), simulate (the genome minus those deletions), test (analyse the *in-silico* cell produced) cycle. The simulations that produce successfully dividing *in-silico* cells would be used to inform the next cycle of simulations. The aim would be to increase the number of gene deletions in each subsequent simulation cycle (while maintaining cellular division), thereby producing a smaller and smaller genome. This approach eventually produced the published Minesweeper algorithm [56].

4.3.2 Initial Development

The first stage of an early version of the algorithm (Figure 8) took a list of potential gene knockouts (singly non-essential, based on three replicate simulations of each *in-silico* single gene knockout) and divided them into groups of deletions of different sizes. This produced an *in-silico* cell that deleted 72 genes and still divided. An *in-silico* cell that removed all 144 singly non-essential genes produced a metabolic mutant phenotype and failed to divide. This was expected due to the likelihood of redundant essential gene relationships between the deleted genes, resulting in synthetic lethality [12,13,52] (Section 1.3.2.a).

The second stage of this early version of the algorithm combined the 50% (72 gene deletion) with non-overlapping smaller groups of deletions (from the remaining singly non-essential genes) that still produced a dividing cell. An *in-silico* cell that combined the 50% deletion segment with 5 non-overlapping 6.5% (8 gene) segments deleted 132 genes and divided. However, the 132-gene-deletion *in-silico* cell displayed a low phenotypic penetrance (the occurrence of particular phenotype given a specific genotype), as it was only producing a successfully dividing cell in 54% of simulation attempts.

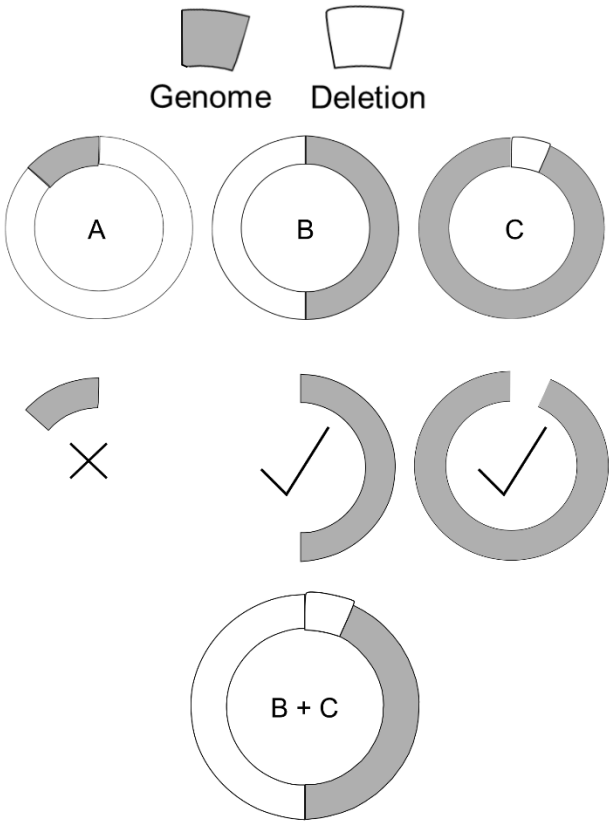

Figure 8. An early version of stage 1 and 2. In the first stage, B and C are gene deletion simulations that produce dividing *in-silico* cells, whereas A does not. In the second stage, B and C deletion groups are combined, deleted, and simulated.

4.3.3 Investigating Deletions

The impact of particular gene deletions on phenotypic penetrance was investigated *in-silico*. Previously a group of genes had been identified, from repeated single gene knockouts, as not having a high enough phenotypic penetrance to be reliably considered non-essential (i.e. they produced an *in-silico* dividing cell on occasion, but not every time). Removing 8 of these genes from the 132 deletion group, resulted in a 124 gene deletion set where the resulting cell was able to divide 86% of the time.

At this stage, I repeated the single gene knockout simulations and increased the number of repetitions to 10 in an attempt to clarify some of the gene knockouts presenting different phenotypes or having low phenotypic penetrance. 19 additional genes (Table 7, Section 4.4.1) were reclassified as singly non-essential, as defined in Section 1.3.2.a, and determined by binominal proportion confidence interval; which

estimates the probability of success for experiments with only two possible outcomes (success / failure), when only the past number of experiments and number of successes are known. It produces two scores which give a range around the number of observed successful experiments, with 95% confidence that the true number of successful experiments lies within this range. For example, a gene knockout produces a dividing cell 9 out of 10 times and generates the score 0.55 and 0.99, which creates the range of 8.45 to 9.99 for containing the true value. As the number of dividing cells has to be a whole number, you can assume that the true number is 9, and make a reclassification decision from there. For the statistical implementation, see Section 2.4.2.

These were iteratively added to the group of 124 deletions, as well as two non-essential genes that had yet to be removed, increasing the number of gene deletions to 145, creating a dividing cell with a 256 gene genome *in-silico*.

4.3.4 Secondary Development

At this point in time, the later stages of the algorithm lacked definitive steps. The third stage continued the second stage using smaller groups of gene deletions (that also produced dividing *in-silico* cells when removed), until no more groups could be added. The fourth stage consisted of cross comparing simulation results manually to identify single gene deletions that were still candidates for gene deletion. However, these stages of the algorithm would have to be codified into a computational algorithm, a necessary step to make the research repeatable and publishable. This prompted me to revisit and formalise these later stages, making them specific and programmable.

This revisit also allowed me to rectify a mistake I originally made in designing the first stage. Due to the failure of the 100% deletion to produce a dividing cell, I made an assumption that anything larger than a 50% deletion segment would not produce a dividing cell, which in retrospect is not defensible. Later research that discovered several low essential genes (Section 4.4.7) allowed me to quantify the impact of this assumption. These low essential genes were between gene positions 82 to 90, roughly in the middle of the original deletion set (144 genes). This positioning prevented deletion

segments larger than 50% from producing a dividing cell *in-silico*, so even with the false assumption, no greater progress could have been produced in stage 1.

4.3.5 Stage 1 Implementation

When codifying and iterating on Minesweeper, the design aim became: tracking, grouping, and combining genes in a comprehensive and failure-proof way; in a transparent manner that allows progress and gene combinations to be checked after the fact; and working within the computational limitations of the BlueGem supercomputer. The verbose writing style of the code, the use of comments, and the lack of external libraries / tools aligns with this transparency and hopefully improves the understanding of the code and design decisions made.

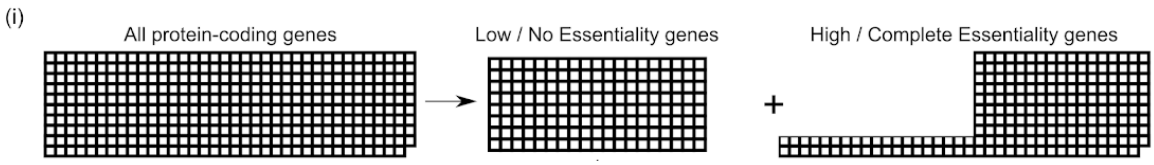

Figure 9. Visualisation of Minesweeper Stage 1

Stage 1 (Figure 9) produces hundreds of single protein-coding (Section 1.3.2.c) gene knockout simulations to identify low and no essentiality genes (whose knockout does not prevent cell division) for Stage 2. It was designed this way to identify complete / high essential genes (Section 1.3.2.a) to exclude from future simulations, in an attempt to keep the number of dividing cells high. The code creates single gene knockout simulation scripts from an initial list of genes.

Functions:

- userInput
    - Ask the user for variables which are required by the supercomputer to run simulations on the users account, which are later used to modify a template script.
- createGeneList
    - Converts a given text file into the correct gene format making it usable. The *M.genitalium* whole-cell model processes genes in the specific

format “ ‘MG_XXX’, ”. This allows the user to work with the whole genome, subsets, or just particular genes.

- createScripts
    - Converts a template script using user input, creating simulation scripts. This automation keeps settings, supercomputer variables, and formatting consistent between simulations.
    - Two pathways depending on the number of genes: <200 genes creates one script, >200 genes creates multiple scripts. The maximum number of simulations a user can start with one script on BlueGem is 256, any further simulations are automatically cancelled. Keeping the threshold at 200 simulations safely avoids this.

4.3.6 Stage 2 Implementation

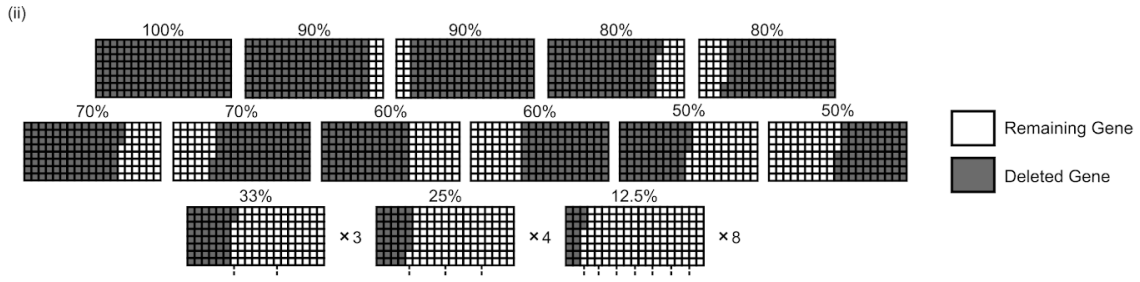

Figure 10. Visualisation of Minesweeper Stage 2

Stage 2 (Figure 10) produces 26 deletion segments to be simulated, ranging in size from 100% to 12.5% of the identified low / no essentiality genes. This is to create the largest possible number of deletions from the *in-silico* genome in one step. The smallest deletion segment size is set by supercomputer limitations, as 12.5% creates 8 groups of gene deletions (Section 2.4.5.6). The code creates deletion segment simulation scripts from the results of Stage 1 simulations (outputted as inputko*_endtimes.txt). Deletion segments that can be removed and still produce a dividing cell are carried forward to Stage 3.

A way to improve this implementation, instead of including smaller deletion segments, could be to implement other variants of the deletion segments. Given that the location

of low essential genes is unknown you could produce variants that split and space the deletions over the genome i.e. the 50% variant could delete the first 25% of the low / no essentiality genes, not delete the next 25%, delete the third 25% of the low / no essentiality genes, and not delete the final 25%. This would increase the number of combinations to match and track but has the potential to be codifiable.

Functions:

- interpretResults
    - Match the simulation results of Stage 1 to the knocked out gene.
- createNEList
    - Filter the Stage 1 matched results, finding those that divided, and labelling and excluding high / complete essentiality genes.
- segmentGeneration
    - Used to select which genes to include in the 26 deletion segments.
- outputToLists
    - Used to save the 26 deletion segments as separate files, and a combined file after the final segment is generated.
- createDivisionSegments
    - Generates the 26 deletion segments, using the functions segmentGeneration and outputToLists.
- createScripts
    - Converts a template script using user input and the combined file produced in this stage, creating simulation scripts for the 26 deletion segments. The same automation is used in all stages.

The use of text files to record information rather than pass the information dynamically to functions is in line with the aim of transparency. It allows the user to track genomic edits, allows the current stage of Minesweeper to locate the information it needs to produce simulation scripts in a known location, and allows future stages of Minesweeper to analyse the results.

4.3.7 Stage 3 Implementation

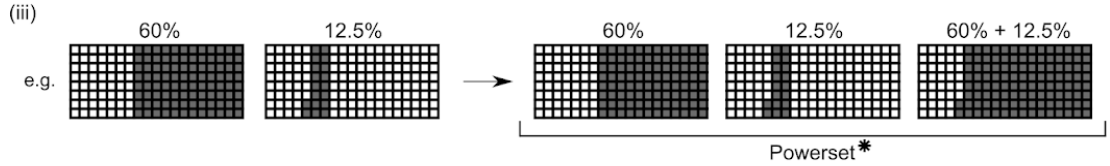

Figure 11. Visualisation of Minesweeper Stage 3. * = a complete powerset also includes a set with zero deletion segments that is not shown here.

Stage 3 (Figure 11) ranks the output of Stage 2, progressing with the largest deletion segment that can be removed and produce a dividing cell, which is matched with other division-producing, non-overlapping deletion segments. This filtering of segments increases the chances of success and lowers the total number of segments to match and combine (see computational limitations, Section 2.4.5.6).
A powerset is then generated (i.e. a set containing all possible unique combinations of the matched deletion segments, including zero and individual deletion segments) and each of the deletion combinations is removed from an individual *in-silico* cell and simulated. This ensures that all possible combinations are simulated. Deletion segments that can be removed and still produce a dividing cell are carried forward to Stage 4.

Functions:

- alreadyRunCheck
    - Checks to see if input files are present and output files have been previously generated, then queries the user. Both Script 3 and 4 use powerset. The output of powerset is the same given the same input, but the order of combinations of the same size is randomised. As matching of the simulation results is by order, this could match results to the wrong simulation if rerun, which would require restarting Minesweeper. This function prevents files previously generated using powersets being changed and overwritten, requiring input from the user before continuing.
- interpretResults
    - Match the simulation results with the 26 deletion segments from Stage 2.

- successCheckAndVariants
    - Check 26 deletion segments for division and save the top three largest deletions. These top three are assigned to colours (red, yellow, blue) and used as three variants / avenues of deletion going forward. This is a check on whether the results are converging, and allows the user to continue simulations if one variant reaches a “dead end” prematurely. Stage 4 should avoid this, but this acts as a failsafe.
- powerset
    - Generates a powerset, all possible unique combinations of a set e.g. 1,2,3 = () (1,) (2,) (3,) (1,2) (1,3) (2,3) (1,2,3).
- outputToLists
    - Uses powerset to generate combinations. Records the powerset combinations and identifies genes to be deleted.
- variantCombinations
    - The variants (three largest deletion segments) are matched with all other dividing, non-overlapping segments, using combinational logic (outlined below).
- createScripts
    - Converts a template script using user input and the powerset combinations, creating simulation scripts.

The combinational logic is based on visualising the genome linearly from left to right, breaking it down into different size segments, and then matching those segments that do not overlap or overlap only by a few percent (Figure 12). The model does not penalise deleting the same gene multiple times, so slight overlap in the deleted powerset combinations ensures coverage of the genome while keeping the number of matches low.

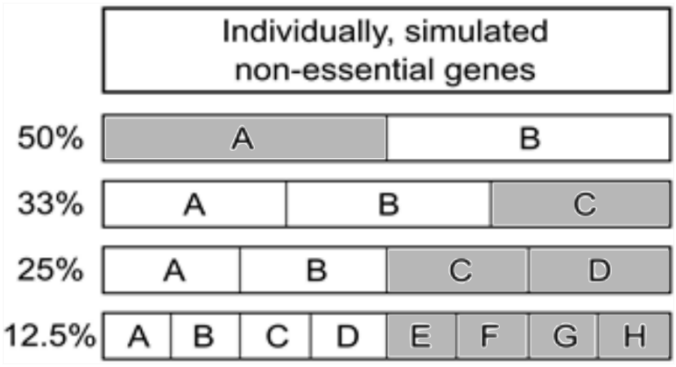

Figure 12. Combinational logic example. If the 50%a deletion produces a dividing *in-silico* cell, it could be matched with 33%c, 25%c, 25%d, and 12.5% e - h if they also produce dividing *in-silico* cells, as these do not overlap. When potential matches duplicate genes (25%c - d vs 12.5%e - h), only the smaller segments are selected for combination, reducing the overall number of segments to combine.

- If 100 % of low / no essentiality genes are deleted: Minesweeper finishes.
- If 90% of low / no essentiality genes are deleted: Move onto Stage 4.
- If 80% of low / no essentiality genes are deleted, potential matches for the A variant are: '12.5% g', '12.5% h'
- If 70% of low / no essentiality genes are deleted, potential matches for the A variant are: '12.5% f', '12.5% g', '12.5% h'
- If 60% of low / no essentiality genes are deleted, potential matches for the A variant are: '33% c', '12.5% e', '12.5% f', '12.5% g', '12.5% h'
- If 50% of low / no essentiality genes are deleted, potential matches for the A variant are: '33% c', '12.5% e', '12.5% f', '12.5% g', '12.5% h'
- If 33% of low / no essentiality genes are deleted, potential matches for the A variant are: '33% b', '33% c', '12.5% d', '12.5% e', '12.5% f', '12.5% g', '12.5% h'
- If 25% of low / no essentiality genes are deleted, potential matches for the A variant are: '12.5% c', '12.5% d', '12.5% e', '12.5% f', '12.5% g', '12.5% h'
- If 12.5% of low / no essentiality genes are deleted, potential matches for the A variant are: '12.5% b', '12.5% c', '12.5% d', '12.5% e', '12.5% f', '12.5% g', '12.5% h'

4.3.8 Stage 4 Implementation

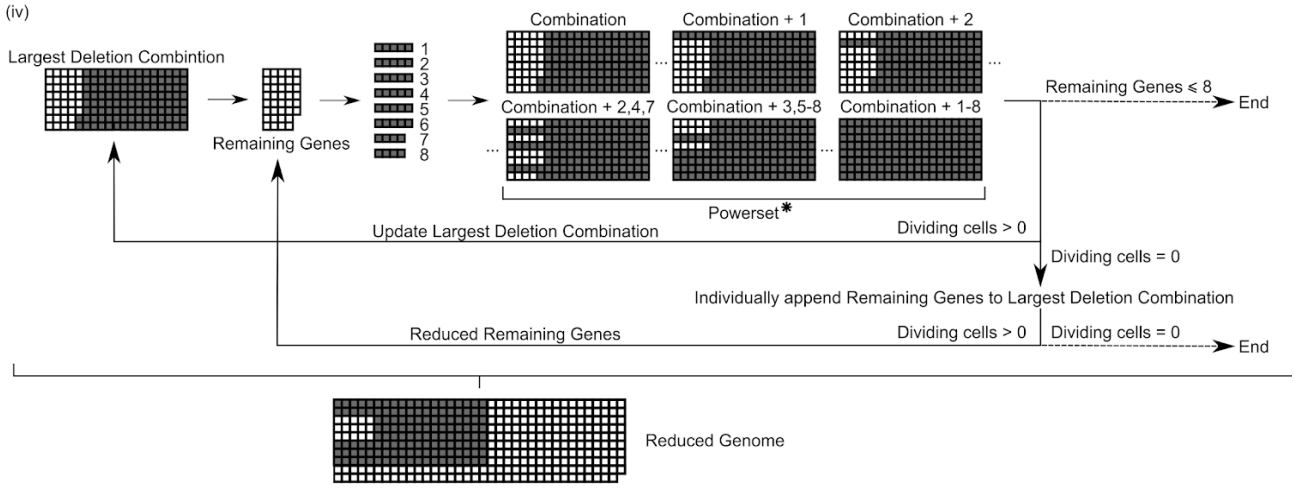

Figure 13. Visualisation of Minesweeper Stage 4. * = a complete powerset also includes a set with zero deletion segments that is not shown here.

Stage 4 (Figure 13) is cyclical. It ranks the results from Stage 3 and identifies the largest deletion segment to determine the remaining low / no essentiality genes that have not been deleted. This is done for each variant. The remaining genes are divided into eight groups (the number of groups is determined by computational constraints, Section 2.4.5.6) and a powerset generated for these eight groups. Each combination from the powerset is individually appended to the current largest deletion combination for simulation. The simulation results update the largest deletion combination, which is used to generate a new remaining gene list, starting the next cycle.
If none of the simulations produces a dividing cell, the remaining genes are singly appended to the prior largest deletion combination, removed and simulated. The individual remaining genes that do not produce a dividing cell are excluded for a cycle, a reduced remaining gene list is produced and used for the subsequent round of Stage 4.
Stage 4 continues until there are eight or fewer remaining genes (where a final appended powerset is run) or all individually appended remaining genes do not produce a dividing cell. Both outcomes result in a list of deleted genes and identified low essential genes.

Functions:

- alreadyRunCheck
    - Checks to see if input files are present and output files have been generated, then queries the user.
- interpretResults
    - Match the simulation results of Stage 3 / Stage 4 to the specific powerset combination.
- remainingGenes
    - Calculate remaining genes from dividing results. Three outcomes:
        - If there is no division, return to prior round's remaining genes
        - If prior round contained appended single knockouts, create a reduced remaining gene set from those that divided
        - If there is division, rank results and select the smallest number of remaining genes
- powerset
    - Generates a powerset, all possible unique combinations of a set e.g. 1,2,3 = () (1,) (2,) (3,) (1,2) (1,3) (2,3) (1,2,3).
- outputToLists
    - Uses powerset to generate combinations. Records the names of eight groups in the combinations, and matches with the genes within the group.

The following two functions are unique to Stage 4 and deal with splitting the remaining genes into eight groups.

- eightsegmentwrite
    - Outputs remaining gene lists for each of the eight groups.
- eightPanelGroupingsGeneration
    - Depending on the number of remaining genes, the size of the group changes. Groups one to seven are equal in size, with group eight differing in size depending on if the number of remaining genes are even or odd. Three possible options:

        - 15 genes or greater = divides the number of genes into the eight groups (minimum two, apart from final/eighth group) and allocates them.
        - 14 genes to 8 genes = a hard coded number of genes selected, maintaining eight groups by transitioning the minimum number of genes per group from two to one. This was created to prevent errors where 1 to 4 groups had no genes allocated, as the minimum gene number for groups was previously set at two.
        - 7 genes to 1 genes = number of groups is equal to the number of genes, with each group containing one gene.
- runappendsingleKOS
    - Single gene knockouts are appended to the largest deletion with a successful dividing cell, to be simulated.
- createScripts
    - Converts a template script using user input, creating simulation scripts.
- endingDecision
    - Given the outcome of createDividingTxt and remainingGenes, determine next step for remaining variants (e.g. Red, Yellow, Blue).
    - If the number of remaining genes was less than or equal to eight in the last simulation round (triggering the final round), record the final results of the variant. If all three variants are complete, finish Minesweeper.
    - If single gene knockouts of remaining genes individually appended to largest deletion were conducted in the last simulation round and none divided, record final results.
    - If a normal eight group powerset was conducted in the last simulation round, and none divided, start an appended single gene knockouts round.
    - If a normal eight group powerset was conducted in the last simulation round, and some divided, continue to another round or the final round depending on if remaining genes are less than or equal to eight genes.

4.4 Results

4.4.1 Initial Input

To generate an initial input for Minesweeper and GAMA I simulated single gene knockouts in an unmodified *M.genitalium in-silico* genome (as previously reported [6], Table 6). Of the 401 *in-silico* modelled genes 42 are RNA-coding, which were not selected for knockout. Gene knockouts in each of the 359 protein-coding genes were simulated individually (10 replicates each), with 152 genes classified as non-essential and 207 genes classified as essential (i.e. producing a dividing or non-dividing *in-silico* cell, respectively).

Gene	Decision	Gene	Decision	Gene	Decision	Gene	Decision	Gene	Decision
MG_001	No Division	MG_051	No Division	MG_100	No Division	MG_156	No Division	MG_198	No Division
MG_003	No Division	MG_052	Dividing	MG_101	Dividing	MG_157	No Division	MG_200	Dividing
MG_004	No Division	MG_053	No Division	MG_102	No Division	MG_158	No Division	MG_201	No Division
MG_005	No Division	MG_055	Dividing	MG_476	Dividing	MG_159	No Division	MG_203	No Division
MG_006	No Division	MG_473	No Division	MG_104	Dividing	MG_160	No Division	MG_204	No Division
MG_007	No Division	MG_058	No Division	MG_105	Dividing	MG_161	No Division	MG_205	Dividing
MG_008	No Division	MG_059	Dividing	MG_106	No Division	MG_162	No Division	MG_206	Dividing
MG_009	Dividing	MG_061	Dividing	MG_107	No Division	MG_163	No Division	MG_208	Dividing
MG_012	Dividing	MG_062	Dividing	MG_109	Dividing	MG_164	No Division	MG_209	Dividing
MG_013	No Division	MG_063	Dividing	MG_110	Dividing	MG_165	No Division	MG_210	Dividing
MG_014	Dividing	MG_064	Dividing	MG_111	No Division	MG_166	No Division	MG_481	No Division
MG_015	Dividing	MG_065	Dividing	MG_112	No Division	MG_167	No Division	MG_482	Dividing
MG_019	No Division	MG_066	No Division	MG_113	No Division	MG_168	No Division	MG_212	No Division
MG_020	Dividing	MG_069	No Division	MG_114	No Division	MG_169	No Division	MG_213	Dividing
MG_021	No Division	MG_070	No Division	MG_118	No Division	MG_170	Dividing	MG_214	Dividing
MG_022	No Division	MG_071	No Division	MG_119	Dividing	MG_171	No Division	MG_215	No Division
MG_023	No Division	MG_072	Dividing	MG_120	Dividing	MG_172	Dividing	MG_216	No Division
MG_026	No Division	MG_073	Dividing	MG_121	Dividing	MG_173	No Division	MG_217	Dividing
MG_027	Dividing	MG_075	Dividing	MG_122	Dividing	MG_174	No Division	MG_218	Dividing
MG_029	Dividing	MG_077	No Division	MG_123	Dividing	MG_175	No Division	MG_221	No Division
MG_030	Dividing	MG_078	No Division	MG_124	No Division	MG_176	No Division	MG_224	No Division
MG_031	No Division	MG_079	No Division	MG_126	No Division	MG_177	No Division	MG_225	Dividing
MG_033	Dividing	MG_080	No Division	MG_127	Dividing	MG_178	No Division	MG_226	Dividing
MG_034	No Division	MG_081	No Division	MG_128	No Division	MG_179	No Division	MG_227	Dividing
MG_035	No Division	MG_082	No Division	MG_130	Dividing	MG_180	No Division	MG_228	No Division
MG_036	No Division	MG_083	Dividing	MG_132	Dividing	MG_181	No Division	MG_229	No Division
MG_037	No Division	MG_084	No Division	MG_136	No Division	MG_182	No Division	MG_230	No Division
MG_038	No Division	MG_085	Dividing	MG_137	No Division	MG_183	Dividing	MG_231	No Division
MG_039	Dividing	MG_086	Dividing	MG_139	Dividing	MG_184	Dividing	MG_232	No Division
MG_040	Dividing	MG_087	No Division	MG_141	No Division	MG_186	Dividing	MG_234	No Division
MG_041	No Division	MG_088	No Division	MG_142	No Division	MG_187	Dividing	MG_235	Dividing
MG_042	No Division	MG_089	No Division	MG_143	Dividing	MG_188	Dividing	MG_236	Dividing
MG_043	No Division	MG_090	No Division	MG_145	No Division	MG_189	Dividing	MG_238	No Division
MG_044	No Division	MG_091	No Division	MG_149	Dividing	MG_190	Dividing	MG_239	Dividing
MG_045	No Division	MG_092	No Division	MG_150	No Division	MG_191	Dividing	MG_240	Dividing
MG_046	Dividing	MG_093	No Division	MG_151	No Division	MG_192	Dividing	MG_244	Dividing
MG_047	No Division	MG_094	No Division	MG_152	No Division	MG_194	No Division	MG_245	No Division
MG_048	Dividing	MG_097	Dividing	MG_153	No Division	MG_195	No Division	MG_249	No Division
MG_049	No Division	MG_098	No Division	MG_154	No Division	MG_196	No Division	MG_250	No Division
MG_050	Dividing	MG_099	No Division	MG_155	No Division	MG_197	No Division	MG_251	No Division

Gene	Decision	Gene	Decision	Gene	Decision	Gene	Decision
MG_252	Dividing	MG_309	Dividing	MG_363	No Division	MG_418	No Division
MG_253	No Division	MG_310	Dividing	MG_522	No Division	MG_419	No Division
MG_254	No Division	MG_311	No Division	MG_365	No Division	MG_421	Dividing
MG_257	No Division	MG_312	Dividing	MG_367	No Division	MG_424	No Division
MG_258	No Division	MG_315	No Division	MG_368	No Division	MG_425	Dividing
MG_259	Dividing	MG_316	Dividing	MG_369	Dividing	MG_426	No Division
MG_261	No Division	MG_317	Dividing	MG_370	Dividing	MG_427	Dividing
MG_262	Dividing	MG_318	Dividing	MG_372	No Division	MG_428	Dividing
MG_498	Dividing	MG_321	No Division	MG_375	No Division	MG_429	No Division
MG_264	Dividing	MG_322	No Division	MG_376	Dividing	MG_430	No Division
MG_265	Dividing	MG_323	No Division	MG_378	No Division	MG_431	No Division
MG_266	No Division	MG_324	Dividing	MG_379	No Division	MG_433	No Division
MG_270	No Division	MG_325	No Division	MG_380	Dividing	MG_434	No Division
MG_271	No Division	MG_327	Dividing	MG_382	No Division	MG_435	No Division
MG_272	No Division	MG_329	Dividing	MG_383	No Division	MG_437	No Division
MG_273	No Division	MG_330	No Division	MG_384	No Division	MG_438	Dividing
MG_274	No Division	MG_333	Dividing	MG_385	Dividing	MG_442	Dividing
MG_275	No Division	MG_334	No Division	MG_386	Dividing	MG_444	No Division
MG_276	No Division	MG_335	Dividing	MG_387	No Division	MG_445	No Division
MG_277	Dividing	MG_517	No Division	MG_390	Dividing	MG_446	No Division
MG_278	No Division	MG_336	Dividing	MG_391	Dividing	MG_447	Dividing
MG_282	No Division	MG_339	Dividing	MG_392	Dividing	MG_448	Dividing
MG_283	No Division	MG_340	No Division	MG_393	Dividing	MG_451	No Division
MG_287	No Division	MG_341	No Division	MG_394	No Division	MG_453	No Division
MG_288	Dividing	MG_342	No Division	MG_396	No Division	MG_454	Dividing
MG_289	Dividing	MG_344	Dividing	MG_398	Dividing	MG_455	No Division
MG_290	Dividing	MG_345	No Division	MG_399	Dividing	MG_457	Dividing
MG_291	Dividing	MG_346	Dividing	MG_400	Dividing	MG_458	No Division
MG_292	No Division	MG_347	No Division	MG_401	Dividing	MG_460	Dividing
MG_293	Dividing	MG_349	Dividing	MG_402	Dividing	MG_462	No Division
MG_295	No Division	MG_351	No Division	MG_403	Dividing	MG_463	Dividing
MG_297	Dividing	MG_352	Dividing	MG_404	Dividing	MG_464	Dividing
MG_298	Dividing	MG_353	Dividing	MG_405	Dividing	MG_465	No Division
MG_299	No Division	MG_355	Dividing	MG_407	No Division	MG_466	No Division
MG_300	No Division	MG_356	Dividing	MG_408	Dividing	MG_467	Dividing
MG_301	No Division	MG_357	No Division	MG_409	Dividing	MG_468	Dividing
MG_302	No Division	MG_358	Dividing	MG_410	Dividing	MG_526	Dividing
MG_303	No Division	MG_359	Dividing	MG_411	Dividing	MG_469	No Division
MG_304	No Division	MG_361	No Division	MG_412	Dividing	MG_470	No Division
MG_305	Dividing	MG_362	No Division	MG_417	No Division		

Table 6. Simulating protein-coding *M.genitalium* single gene knockouts. A published version of the table is available [56], as is the simulation data [161].

The majority of genes (58%) are essential; this was expected, as *M.genitalium* is an obligate parasite with reduced genetic redundancy [61]. 318 genes showed consistent phenotype across replicates, with 41 showing inconsistent phenotypes. Statistical analysis (binomial proportion confidence interval, Pearson-Klopper, 95% CIs for: a 6/10 replicate [5.74, 6.87], 7/10 replicates [6.66, 7.93], 8/10 replicates [7.56, 8.97], 9/10 replicates [8.45, 9.99]) resulted in assigning the most common phenotype (Section 2.4.2 and Table 7).

Overall, my results agree 97% with Karr *et al.* [6] (Table 8), disagreeing on nine of the 359 protein-coding genes. From reading Karr *et al.*'s Supplementary Information [6] and investigating the model's code base, I believe that the original classifications were based on five simulations per gene, with the results then averaged together, and a classification assigned computationally to the averaged data. By comparison, I completed ten simulations per gene and assigned classifications manually. Our differences may be due to edge cases that were misclassified by averaging and computational assessment.

Gene	Phenotypes	Consistency	Decision
MG_084	Protein Mutant / Dividing	6 out of 10	No Division
MG_387	Septum Mutant / Slow Growing Mutant Mutant / Dividing	7 out of 10	No Division
MG_049	Protein Mutant / Septum Mutant / Dividing	7 out of 10	No Division
MG_250	DNA Mutant / Slow Growing Mutant / Dividing	8 out of 10	No Division
MG_470	Septum Mutant / Dividing	8 out of 10	No Division
MG_384	Septum Mutant / Dividing	8 out of 10	No Division
MG_008	Protein Mutant / Dividing	8 out of 9	No Division
MG_379	Protein Mutant / Dividing	9 out of 10	No Division
MG_445	Protein Mutant / Dividing	9 out of 10	No Division
MG_126	Protein Mutant / Dividing	9 out of 10	No Division
MG_295	Protein Mutant / Dividing	9 out of 10	No Division
MG_122	Dividing / Septum Mutant	7 out of 10	Dividing
MG_104	Dividing / Metabolic Mutant / Slow Growing Mutant / DNA Mutant	7 out of 10	Dividing
MG_438	Dividing / DNA Mutant	8 out of 10	Dividing
MG_425	Dividing / DNA Mutant	8 out of 9	Dividing
MG_410	Dividing / DNA Mutant / Slow Growing Mutant	9 out of 10	Dividing
MG_289	Dividing / DNA Mutant	9 out of 10	Dividing
MG_324	Dividing / DNA Mutant	9 out of 10	Dividing
MG_352	Dividing / DNA Mutant	9 out of 10	Dividing
MG_359	Dividing / DNA Mutant	9 out of 10	Dividing
MG_052	Dividing / DNA Mutant	9 out of 10	Dividing
MG_401	Dividing / DNA Mutant	9 out of 10	Dividing
MG_464	Dividing / DNA Mutant	9 out of 10	Dividing
MG_127	Dividing / DNA Mutant	9 out of 10	Dividing
MG_183	Dividing / DNA Mutant	9 out of 10	Dividing
MG_189	Dividing / DNA Mutant	9 out of 10	Dividing
MG_190	Dividing / DNA Mutant	9 out of 10	Dividing
MG_214	Dividing / DNA Mutant	9 out of 10	Dividing
MG_236	Dividing / DNA Mutant	9 out of 10	Dividing
MG_265	Dividing / DNA Mutant	9 out of 10	Dividing

Table 7. Inspection of 41 genes that produced inconsistent phenotypes. A published version of the table is available [56], as is the simulation data [161].

Gene	Karr *et al.* 2012	My Results
MG_019	No Division over Two Generations*	No Division (Septum Mutation)
MG_049	Dividing	No Division (Protein Mutation)
MG_051	Dividing	No Division (Protein Mutation)
MG_084	Dividing	No Division (Protein Mutation)
MG_122	No Division	Dividing
MG_126	Dividing	No Division (Protein Mutation)
MG_203	Dividing	No Division (Septum Mutation)
MG_384	No Division over Two Generations*	No Division (Septum Mutation)
MG_470	Dividing	No Division (Septum Mutation)

Table 8. Differences between results and Karr *et al.* 2012. A published version of the table is available [56], as is the simulation data [161]. * = growth sub-model only.

4.4.2 Minesweeper Results

Minesweeper (Sections 4.3.5 to 4.3.8) produced results quickly; within two days the third stage removed 123 genes (a 34% reduction), an equivalent size reduction to current lab-based efforts in other species [22,86,87]. In total, Minesweeper deleted 145 genes, creating an *in-silico M.genitalium* cell containing 256 genes (named Minesweeper_256) and predicting an *in-vivo* minimal genome of 380 genes. The *in-silico* cell replicates DNA, produces RNA and protein, grows, and divides.

4.4.3 GAMA Method and Results

This work is presented in full in Oliver Chalkley's thesis and is available in published papers [56,164]. In summary, GAMA is a biased genetic algorithm [166], conducting two stages (Guess and Add) of only non-essential gene deletions, before including essential genes for deletion in the third stage (Mate). The smallest GAMA-reduced *in-silico* genome deleted 165 genes, creating an *in-silico M.genitalium* genome of 236 genes (named GAMA_236), and predicting an *in-vivo* minimal genome of 360 genes. GAMA removed more genes than Minesweeper, while still producing a simulated cell which replicates DNA, produces RNA and protein, grows, and divides.

4.4.4 Comparison of GAMA and Minesweeper

Minesweeper and GAMA conduct whole-cell model simulations using the same three step design-simulate-test cycle (Section 4.3.1).

Minesweeper has generated 4620 *in-silico* genomes, which means it is still feasible for each genome produced to be manually analysed. It initially deletes genes in groups but eventually deletes individual genes, and only deletes non-essential genes. It produces large gene deletions more quickly. Firstly, as it excludes essential genes from being deleted it can delete a larger number of genes with a lower risk of producing non-dividing *in-silico* cells. Secondly, it uses between 8 and 359 CPUs depending on the stage, processing through the supercomputer queue more quickly. This allows it to produce minimal genome size reductions within two days. As data storage is handled by user-submitted information and simulation execution conducted manually, it only requires a basic understanding of common computational tools (i.e. terminals, FTP software), maintaining the possibility of biologists being able to use it.

GAMA has generated 53,451 *in-silico* genomes, with deletions varying by individual genes. It does however take much longer, with two months required to generate minimal genome size reductions. It requires greater computational power and time, using between 400 and 3000 CPUs depending on the stage, for longer periods of time than is allowed on available supercomputers, requiring the development of a seperate tool (the genome design suite [164]) to implement GAMA. It also requires greater computational knowledge, needing a good knowledge of querying SQL databases, and the capacity to install and learn to use the genome design suite [164].

In comparing the two algorithms, GAMA is more comprehensive. It produces greater numbers of simulations that cover a greater amount of the solution space. It also attempts to remove singly essential genes, which are the key differences between the smallest genomes produced by Minesweeper and GAMA. The size of the genome database produced by GAMA means that machine learning approaches can be applied, which is attractive to other research disciplines. Further research may be able to identify other minima that have yet to be discovered in the database (the number of

genomes produced is too great to manually analyse) and identify key features shared by the 10,000s of genomes stored there. The genome design suite tool may also be adaptable to other genome optimisation problems, but that requires new researchers to learn how to use the tool and further develop it by themselves.

Minesweeper is the solution to a specific problem, which it executes on quickly, and could be used by other biologists for new whole-cell models. GAMA, the genome dataset, and the genome design suite are worthy of continued development, analysis, and research, as they could be key to solving multiple research problems in the future.

4.4.5 Minesweeper_256, GAMA_236 and GAMA_237 Genomes

I investigated our two minimal genomes for their consistency in producing a dividing *in-silico* cell, and the range of behaviour they displayed. I simulated 100 replicates each of an unmodified *M.genitalium in-silico* genome, Minesweeper_256, GAMA_236, and a single gene knockout of a known essential gene (MG_006) to provide a comparison (Table 9). The rate of division (or lack of in the MG_006 knockout simulations) was analysed and a phenotype penetrance percentage calculated to quantify how often an expected phenotype occurred. The unmodified *M.genitalium* and MG_006 knockout *in-silico* genomes demonstrated consistent phenotypes (99% and 0% divided, respectively). Minesweeper_256 was slightly less consistent (89% divided), while GAMA_236 was substantially less consistent, producing a dividing *in-silico* cell 18% of the time. This is not entirely unexpected given the presence of gene deletions that have high essentiality (Section 4.4.8, Table 19). I attempted to improve the division rate of GAMA_236 by conducting independent single knock-ins of its unique deletions (Table 10). The highest division rate was 33% (100 replicates, Table 9) due to the reintroduction of a single gene (MG_270), creating the *in-silico* minimal genome GAMA_237. MG_270 (lipoate-protein ligase A) modifies the enzyme produced by MG_272 (Karr *et al.* Supplementary mmc1, pg 65) [6], one of four genes (MG_271-274) that produce enzymes that form the pyruvate dehydrogenase complex. This provides acetyl-CoA for the Krebs cycle, producing ATP. Reintroducing MG_270 repairs this complex and increases the available energy in the *in-silico* cell.

	whole-cell model	MG_006 deletion	Minesweeper_ 256	GAMA_236	GAMA_237
Phenotypic Penetrance	99%	100%	89%	18.5%	33%
1	Dividing	-	Dividing	Dividing	No Division
2	Dividing	-	Dividing	Dividing	No Division
3	Dividing	-	Dividing	No Division	No Division
4	Dividing	-	Dividing	No Division	No Division
5	Dividing	-	Dividing	No Division	No Division
6	Dividing	-	Dividing	No Division	No Division
7	Dividing	-	Dividing	Dividing	No Division
8	Dividing	-	Dividing	Dividing	No Division
9	Dividing	-	No Division	No Division	Dividing
10	Dividing	No Division	Dividing	Dividing	No Division
11	Dividing	No Division	Dividing	No Division	Dividing
12	Dividing	No Division	Dividing	No Division	Dividing
13	Dividing	No Division	Dividing	No Division	No Division
14	Dividing	No Division	Dividing	No Division	Dividing
15	Dividing	No Division	No Division	No Division	No Division
16	Dividing	No Division	Dividing	No Division	No Division
17	Dividing	No Division	Dividing	No Division	No Division
18	Dividing	No Division	Dividing	No Division	No Division
19	Dividing	No Division	Dividing	Dividing	Dividing
20	Dividing	No Division	Dividing	No Division	No Division
21	Dividing	No Division	Dividing	No Division	Dividing
22	Dividing	No Division	Dividing	No Division	No Division
23	Dividing	No Division	Dividing	No Division	Dividing
24	Dividing	No Division	Dividing	No Division	No Division
25	Dividing	No Division	Dividing	No Division	No Division
26	Dividing	No Division	Dividing	Dividing	No Division
27	Dividing	No Division	Dividing	No Division	Dividing
28	Dividing	No Division	Dividing	No Division	No Division
29	Dividing	No Division	Dividing	No Division	No Division
30	Dividing	No Division	No Division	Dividing	No Division
31	Dividing	No Division	Dividing	No Division	No Division
32	Dividing	No Division	Dividing	No Division	Dividing
33	Dividing	No Division	Dividing	No Division	No Division
34	Dividing	No Division	Dividing	Dividing	Dividing
35	Dividing	No Division	Dividing	No Division	No Division
36	Dividing	No Division	Dividing	Dividing	No Division
37	Dividing	No Division	Dividing	No Division	No Division
38	Dividing	No Division	Dividing	No Division	No Division
39	Dividing	No Division	No Division	No Division	No Division
40	Dividing	No Division	Dividing	Dividing	Dividing
41	Dividing	No Division	Dividing	No Division	No Division
42	Dividing	No Division	Dividing	No Division	Dividing

	whole-cell model	MG_006 / Mutant	Minesweeper_ 256	GAMA_236	GAMA_237
43	Dividing	No Division	Dividing	No Division	No Division
44	Dividing	No Division	Dividing	No Division	Dividing
45	Dividing	No Division	Dividing	No Division	Dividing
46	Dividing	No Division	No Division	No Division	Dividing
47	Dividing	No Division	Dividing	No Division	No Division
48	Dividing	No Division	Dividing	No Division	No Division
49	Dividing	No Division	Dividing	No Division	No Division
50	Dividing	No Division	Dividing	No Division	Dividing
51	Dividing	No Division	Dividing	No Division	Dividing
52	Dividing	No Division	Dividing	No Division	Dividing
53	Dividing	No Division	Dividing	Dividing	No Division
54	Dividing	No Division	Dividing	No Division	No Division
55	Dividing	No Division	Dividing	Dividing	No Division
56	Dividing	No Division	Dividing	No Division	Dividing
57	Dividing	No Division	Dividing	No Division	No Division
58	Dividing	No Division	Dividing	No Division	No Division
59	Dividing	No Division	Dividing	No Division	Dividing
60	Dividing	No Division	Dividing	No Division	No Division
61	Dividing	No Division	Dividing	No Division	No Division
62	Dividing	No Division	Dividing	No Division	Dividing
63	Dividing	No Division	Dividing	No Division	No Division
64	Dividing	No Division	Dividing	No Division	Dividing
65	Dividing	No Division	No Division	No Division	No Division
66	Dividing	No Division	Dividing	Dividing	Dividing
67	Dividing	No Division	Dividing	No Division	No Division
68	Dividing	No Division	Dividing	No Division	No Division
69	Dividing	No Division	Dividing	No Division	No Division
70	Dividing	No Division	No Division	Dividing	No Division
71	Dividing	No Division	No Division	No Division	No Division
72	Dividing	No Division	Dividing	No Division	No Division
73	Dividing	No Division	Dividing	No Division	Dividing
74	Dividing	No Division	Dividing	No Division	Dividing
75	Dividing	No Division	Dividing	No Division	No Division
76	Dividing	No Division	No Division	No Division	No Division
77	Dividing	No Division	Dividing	No Division	Dividing
78	Dividing	No Division	Dividing	No Division	Dividing
79	Dividing	No Division	Dividing	No Division	No Division
80	Dividing	No Division	Dividing	Dividing	No Division
81	Dividing	No Division	Dividing	No Division	No Division
82	Dividing	No Division	Dividing	No Division	No Division
83	Dividing	No Division	Dividing	No Division	Dividing
84	Dividing	No Division	Dividing	No Division	No Division
85	Dividing	No Division	Dividing	No Division	Dividing
86	Dividing	No Division	Dividing	No Division	No Division

	whole-cell model	MG_006 / Mutant	Minesweeper_256	GAMA_236	GAMA_237
87	Dividing	No Division	Dividing	No Division	No Division
88	Dividing	No Division	Dividing	Dividing	Dividing
89	Dividing	No Division	Dividing	No Division	No Division
90	Dividing	No Division	No Division	No Division	No Division
91	Dividing	No Division	Dividing	No Division	No Division
92	Dividing	No Division	Dividing	Dividing	No Division
93	Dividing	No Division	No Division	No Division	No Division
94	Dividing	No Division	Dividing	No Division	No Division
95	Dividing	No Division	Dividing	No Division	No Division
96	Dividing	No Division	Dividing	No Division	Dividing
97	Dividing	No Division	Dividing	No Division	No Division
98	Dividing	No Division	Dividing	-	Dividing
99	Dividing	No Division	Dividing	-	Dividing
100	No Division	-	Dividing	-	No Division

Table 9. Phenotypic penetrance of an unmodified *M.genitalium* genome, MG_006 deletion (known essential gene), Minesweeper_256, GAMA_236, and GAMA_237. A published version of the table is available [56], as is the simulation data [161]. - = a failed simulation due to supercomputer error.

MG_	008	022	084	141	177	182	249	270	282	295	340	341	347	367	372	379	445	465
%	20	19	17	22	28	21	25	33	16	23	18	15	27	13	22	18	16	19
1	D	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	D
2	ND	ND	D	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND
3	ND	D	ND	ND	ND	D	ND	ND	ND	D	ND	ND	D	ND	ND	ND	ND	ND
4	ND	D	ND	ND	D	D	ND	ND	ND	ND	ND	D	ND	ND	ND	D	ND	ND
5	ND	ND	ND	ND	ND	ND	ND	ND	D	ND	ND	ND	ND	ND	ND	ND	ND	ND
6	D	ND	ND	D	ND	ND	D	ND	ND	D	ND	ND	ND	ND	ND	D	ND	ND
7	ND	ND	ND	ND	D	ND	ND	ND	ND	D	ND	ND	ND	ND	ND	ND	D	D
8	ND	ND	ND	D	ND	D	D	ND	ND	ND	D	ND	ND	ND	ND	ND	ND	ND
9	ND	ND	ND	ND	ND	D	D	D	ND	ND	ND	ND	ND	ND	D	ND	D	ND
10	ND	ND	ND	D	ND	ND	ND	ND	ND	ND	ND	ND	ND	D	ND	ND	ND	ND
11	ND	ND	ND	ND	ND	ND	D	D	ND	D	ND	ND	ND	ND	ND	ND	ND	D
12	ND	ND	ND	ND	D	D	ND	D	ND	ND	ND	ND	ND	D	D	D	ND	ND
13	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND
14	ND	ND	D	ND	ND	ND	ND	D	ND	ND	ND	ND	ND	ND	ND	ND	ND	D
15	ND	ND	ND	D	ND	ND	D	ND	ND	ND	D	D	D	ND	ND	ND	D	D
16	ND	ND	ND	ND	ND	D	ND	ND	ND	ND	ND	D	D	ND	ND	ND	D	ND
17	ND	ND	D	ND	ND	ND	ND	ND	D	ND	ND	ND	ND	ND	ND	ND	ND	ND
18	ND	D	D	ND	ND	ND	ND	ND	ND	D	D	D	ND	ND	D	D	ND	ND
19	ND	ND	ND	D	D	ND	ND	D	D	ND	ND	ND	ND	ND	ND	ND	ND	ND
20	ND	ND	ND	D	ND	D	ND	ND	ND	ND	ND	ND	ND	ND	ND	D	ND	ND
21	ND	ND	D	ND	ND	ND	D	D	ND	ND	D	ND	ND	D	D	ND	ND	ND
22	D	ND	ND	ND	D	ND	ND	ND	ND	D	ND	ND	ND	ND	ND	ND	ND	ND
23	ND	ND	ND	ND	ND	D	D	D	ND	D	ND	ND	ND	ND	D	ND	D	ND
24	ND	ND	ND	ND	D	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND
25	ND	ND	ND	D	ND	ND	ND	ND	D	ND	ND	ND	ND	ND	ND	ND	ND	ND
26	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	D	D
27	ND	ND	ND	ND	ND	ND	D	D	ND	ND	ND	D	ND	ND	ND	ND	ND	ND
28	D	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	D	ND	D	ND	ND	ND
29	D	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND
30	ND	ND	ND	D	ND	ND	D	ND	ND	D	ND	ND	ND	ND	ND	D	ND	ND
31	ND	ND	ND	ND	ND	ND	D	ND	ND	D	ND	ND	ND	ND	ND	ND	ND	ND
32	ND	ND	ND	ND	D	ND	ND	D	ND	ND	ND	ND	ND	ND	D	ND	ND	ND
33	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	D	ND	ND	D	ND	ND	ND	ND
34	D	ND	ND	ND	D	ND	ND	D	ND	ND	D	ND	ND	ND	ND	D	ND	ND
35	ND	ND	ND	D	ND	ND	ND	ND	ND	ND	ND	ND	ND	D	ND	ND	ND	ND
36	ND	ND	D	ND	ND	D	ND	ND	ND	D	D	ND	ND	ND	ND	ND	ND	ND
37	ND	ND	D	ND	ND	ND	ND	ND	ND	D	ND	ND	ND	ND	ND	D	ND	ND
38	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND
39	ND	ND	ND	ND	D	ND	ND	ND	ND	D	ND	ND	ND	ND	ND	ND	ND	ND
40	D	ND	ND	ND	ND	ND	ND	D	ND	D	ND	ND	D	ND	ND	ND	ND	D
41	ND	ND	ND	D	D	ND	ND	ND	ND	ND	ND	ND	D	D	ND	ND	ND	ND
42	ND	D	ND	D	ND	ND	D	D	ND	ND	ND	ND	ND	D	ND	ND	ND	ND
43	D	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	D	ND	ND	D

MG_	008	022	084	141	177	182	249	270	282	295	340	341	347	367	372	379	445	465
44	ND	D	ND	ND	D	ND	ND	D	ND	ND	D	ND	ND	ND	ND	ND	ND	ND
45	ND	D	D	ND	ND	ND	D	D	D	ND	ND	ND	D	ND	ND	ND	ND	ND
46	ND	D	ND	ND	ND	D	ND	D	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND
47	ND	ND	ND	ND	ND	D	ND	ND	ND	D	ND	ND	D	ND	ND	ND	D	ND
48	ND	ND	ND	ND	ND	ND	D	ND	ND	ND	ND	D	D	ND	ND	ND	ND	ND
49	ND	ND	ND	D	D	ND	ND	ND	ND	ND	ND	ND	D	ND	ND	ND	D	ND
50	D	ND	ND	D	ND	ND	ND	D	ND	ND	D	ND	ND	ND	ND	ND	ND	D
51	ND	ND	ND	ND	ND	ND	ND	D	D	D	ND	ND	D	D	D	ND	ND	ND
52	ND	D	ND	ND	D	ND	ND	D	ND	ND	ND	ND	ND	ND	D	ND	ND	ND
53	ND	ND	ND	ND	D	ND	ND	ND	ND	D	ND	ND	D	ND	D	D	ND	ND
54	ND	ND	ND	ND	ND	ND	D	ND	ND	ND	D	ND	D	ND	D	D	ND	D
55	D	D	ND	ND	D	ND	ND	ND	ND	ND	ND	ND	D	ND	D	ND	ND	ND
56	ND	ND	ND	ND	ND	ND	ND	D	ND	ND	D	ND	D	ND	ND	ND	D	ND
57	D	ND	D	D	ND	ND	ND	ND	D	ND	ND	ND	D	ND	ND	ND	ND	ND
58	ND	ND	D	ND	D	ND	ND	ND	ND	ND	ND	ND	D	ND	ND	ND	ND	D
59	ND	ND	ND	ND	ND	ND	D	D	ND	ND	ND	ND	D	ND	ND	ND	ND	ND
60	D	D	ND	ND	D	ND	ND	ND	D	ND	ND	ND	ND	D	ND	ND	ND	D
61	-	ND	D	ND	D	ND	ND	ND	D	ND	ND	ND	ND	ND	ND	ND	ND	ND
62	D	ND	ND	ND	ND	ND	ND	D	ND	ND	ND	ND	D	ND	ND	ND	D	ND
63	ND	D	ND	ND	ND	ND	D	ND	ND	D	D	ND	ND	ND	ND	ND	D	D
64	ND	ND	ND	D	D	ND	ND	D	ND	ND	ND	D	ND	ND	D	D	ND	ND
65	ND	ND	ND	D	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	D	D	ND	D
66	ND	ND	ND	ND	ND	D	ND	D	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND
67	ND	ND	ND	ND	D	ND	ND	ND	ND	ND	ND	D	ND	ND	ND	ND	ND	ND
68	ND	ND	D	ND	ND	D	D	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND
69	D	ND	ND	ND	ND	D	D	ND	ND	D	ND	ND	ND	ND	D	ND	ND	D
70	D	ND	ND	ND	D	ND	ND	ND	ND	D	ND	D	ND	ND	ND	ND	ND	ND
71	ND	D	ND	ND	ND	D	D	ND	ND	ND	ND	D	ND	ND	ND	ND	ND	D
72	D	D	ND	ND	D	ND	ND	ND	ND	ND	D	ND	ND	ND	ND	ND	D	D
73	ND	ND	ND	ND	ND	ND	ND	D	ND	ND	ND	ND	ND	D	ND	ND	D	ND
74	ND	ND	ND	ND	ND	ND	ND	D	ND	ND	D	ND	ND	ND	ND	ND	ND	ND
75	ND	D	ND	ND	ND	D	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND
76	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	D	ND	ND	D	D
77	ND	D	D	ND	ND	ND	D	D	D	D	ND	D	ND	ND	D	ND	ND	ND
78	D	ND	ND	ND	D	D	ND	D	ND	ND	ND	ND	D	ND	ND	ND	ND	ND
79	ND	ND	ND	D	ND	ND	ND	ND	ND	ND	ND	ND	ND	D	ND	ND	ND	ND
80	ND	ND	ND	ND	ND	ND	D	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND
81	D	ND	D	D	ND	ND	ND	ND	ND	ND	ND	D	D	ND	ND	ND	ND	ND
82	ND	ND	ND	D	ND	ND	ND	ND	D	D	ND	ND	D	ND	ND	ND	D	ND
83	D	ND	ND	D	ND	ND	ND	D	ND	ND	ND	ND	ND	D	D	ND	ND	D
84	ND	D	ND	ND	D	ND	D	ND	ND	ND	ND	ND	ND	ND	ND	D	ND	ND
85	ND	ND	ND	D	ND	ND	ND	D	ND	ND	ND	ND	ND	ND	ND	D	ND	ND
86	D	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	D	D	ND	ND	ND	ND	ND
87	ND	ND	D	ND	ND	D	ND	ND	ND	ND	D	ND	D	ND	D	ND	ND	ND
88	ND	ND	D	ND	D	ND	ND	D	ND	ND	ND	ND	ND	ND	ND	D	ND	ND

MG_	008	022	084	141	177	182	249	270	282	295	340	341	347	367	372	379	445	465
89	ND	D	D	ND	ND	D	ND	ND	D	ND	ND	ND	D	ND	ND	ND	ND	ND
90	ND	D	ND	D	ND	ND	ND	ND	D	ND	ND	ND	D	ND	ND	ND	ND	ND
91	ND	ND	ND	ND	ND	ND	ND	ND	D	ND	ND	ND	ND	ND	ND	D	ND	ND
92	ND	ND	ND	ND	D	ND	ND	ND	ND	D	D	ND	D	ND	D	ND	D	ND
93	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	D	ND	ND	ND	ND	ND	ND
94	ND	ND	ND	ND	ND	D	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND
95	ND	ND	ND	ND	ND	D	D	ND	ND	ND	ND	D	ND	ND	ND	ND	ND	ND
96	ND	ND	ND	ND	ND	ND	D	D	ND	ND	ND	ND	ND	ND	D	ND	ND	ND
97	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	D	ND	ND
98	ND	ND	ND	ND	D	ND	ND	D	D	ND	D	ND	ND	ND	D	D	ND	ND
99	ND	D	ND	ND	D	ND	ND	D	D	D	ND	ND	ND	ND	ND	ND	ND	ND
100	ND	ND	ND	ND	ND	ND	D	ND	ND	ND	D	ND	ND	ND	ND	ND	ND	ND

Table 10. Improving the phenotypic penetrance of GAMA_236 by reintroducing individual genes, creating GAMA_237. ND = No Division, D = Dividing. A published version of the table is available [56], as is the simulation data [161]. - = a failed simulation due to supercomputer error.

While exploring further deletions for Minesweeper_256, another individual gene was found that impacted the division rate. Deleting MG_104 (ribonuclease R) decreased the division rate to a 1/9 of its original value. *M.genitalium* has a very small pool of RNAs (Karr *et al.* Supplementary mmc1, pg 85) [6], relying on ribonucleases to recycle RNAs for other cellular processes. Without ribonuclease R (the only modeled ribonuclease), the RNA decay sub-model cannot function *in-silico*, decreasing the amount of available RNAs.

The 100 replicates for each of the unmodified *M.genitalium* genome, Minesweeper_256, and GAMA_237 were plotted to assess the range of behaviour (Figure 14). The unmodified *M.genitalium* whole-cell model (Figure 14, top row) shows the range of expected behaviour for a dividing cell (in line with previous results [6]). Growth, protein production, and cellular mass increase over time, with the majority of cells dividing within 10 hours (see cell diameter change). RNA production fluctuates but increases over time. DNA replication follows a characteristic shape, with some simulations delaying the initiation of DNA replication past ~9 hours.

By comparison, Minesweeper_256 (Figure 14, middle row) displays slower, and in some cases decreasing, growth over time which is capped at a lower maximum.

Protein and cellular mass are generated more slowly, lower amounts are produced, and some erratic behaviour is present. The range of RNA production is narrower compared to the unmodified *M.genitalium* whole-cell model. DNA replication takes longer and initiation can occur later (at 11 hours). Cell division occurs later, between 8 and 13.9 hours. A number of simulations can be seen failing to replicate DNA and divide.

Compared to the other genomes, GAMA_237 (Figure 14, bottom row) shows a much greater range of growth rates. Some grow as fast as the unmodified genome, some are comparable to Minesweeper_256, and some show very low or decreasing growth (this can also be seen in cellular mass). Observable protein levels appear between 2 and 5 hours, followed by a slower rate of protein production in some simulations. The range of RNA production is reduced and the rate of RNA production is slower. Some simulations replicate DNA at a rate comparable to the unmodified genome, others replicate more slowly, and some do not complete DNA replication. Cell division occurs across a greater range of time (6 - 13.9 hours). A number of simulations showing metabolic defects can be seen. These do not produce any growth and can be seen failing to divide.

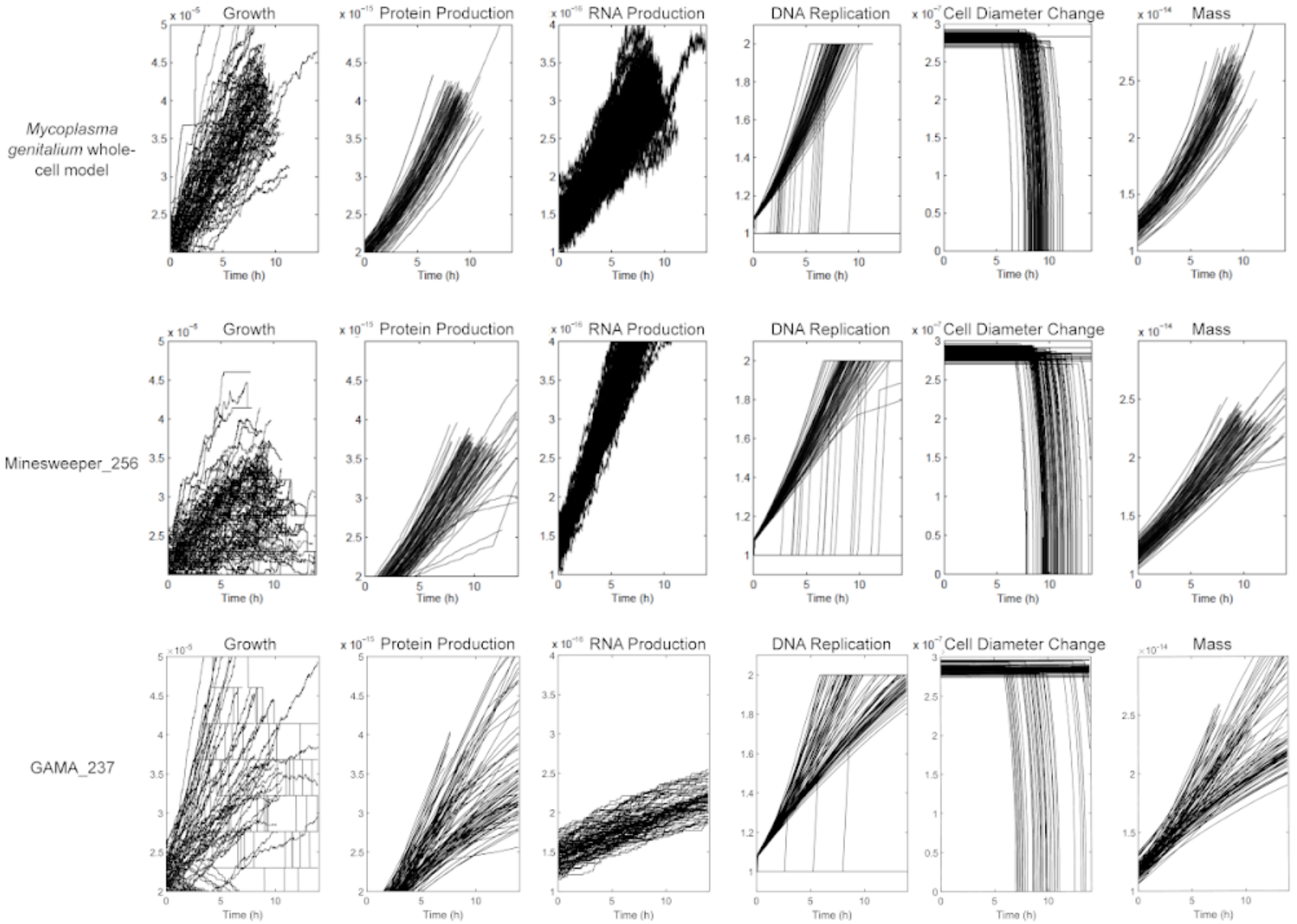
Mycoplasma genitalium whole-cell model
Minesweeper_256
GAMA_237
Growth
Protein Production
RNA Production
DNA Replication
Cell Diameter Change
Mass
Time (h)

Figure 14. Behavioural comparison of the whole-cell model, Minesweeper_256, and GAMA_237. One hundred *in-silico* replicates, with second-by-second values plotted for 6 cellular variables over 13.89 hours (the default endtime of the simulations). The top row shows the expected cellular behaviour (previously shown by Karr *et al.* [6]) and is used for comparison. Minesweeper_256 and GAMA_237 show deviations in phenotype caused by gene deletions. Simulation data is available [161].

4.4.6 Genome Analysis using Gene Ontology terms

I investigated what cellular processes were altered in the creation of Minesweeper_256 using gene ontology (GO) biological process terms [160], standardised labels that describe a gene's function. In the baseline *M.genitalium* whole-cell model 259/401 genes (72% coverage) have GO terms on UniProt [160] (Table 11).

Minesweeper_256 has 186 (73%) genes with GO terms and 70 genes without. The 145 gene deletions reduced 22 (14%) GO categories and removed 42 (27%) GO categories entirely, of which 30 were associated with a single gene (Table 12).

The GO categories reduced include: DNA (repair, replication, topological change, transcription regulation and initiation); protein (folding and transport); RNA processing; creation of lipids; cell cycle; and cell division. As the *in-silico* cells continue to function, we can assume that these categories could withstand low-level disruption.

Removed GO categories that involved multiple genes include: proton transport; host interaction; DNA recombination; protein secretion and targeting to membrane; and response to oxidative stress. Removed GO categories that contain single genes include: transport (carbohydrate, phosphate and protein import, protein insertion into membrane); protein modification (refolding, repair, targeting); chromosome (segregation, separation); biosynthesis (coenzyme A, dTMP, dTTP, lipoprotein); breakdown (deoxyribonucleotide, deoxyribose, mRNA, protein); regulation (phosphate, carbohydrate, and carboxylic acid metabolic processes, cellular phosphate ion homeostasis); cell-cell adhesion; foreign DNA cleavage; SOS response; sister chromatid cohesion; and uracil salvage.

Gene	Name	GO Biological Process Labels
MG_001	dnaN	DNA replication
MG_003	gyrB	DNA topological change
MG_004	gyrA	DNA topological change
MG_005	serS	selenocysteinyl-tRNA(Sec) biosynthetic process, seryl-tRNA aminoacylation
MG_006	tmk	dTDP biosynthetic process
MG_008	mnmE	tRNA modification
MG_012		cellular protein modification process
MG_013	folD	histidine biosynthetic process, methionine biosynthetic process, purine nucleotide biosynthetic process
MG_019	dnaJ	DNA replication, protein folding, response to heat
MG_021	metG	methionyl-tRNA aminoacylation
MG_022	rpoE	regulation of transcription, DNA-templated, transcription, DNA-templated
MG_023	fba	fructose 1, 6-bisphosphate metabolic process, glycolytic process
MG_030	upp	UMP salvage, uracil salvage
MG_031	polC	DNA replication
MG_033	glpF	glycerol metabolic process
MG_034	tdk	DNA biosynthetic process
MG_035	hisS	histidyl-tRNA aminoacylation
MG_036	aspS	tRNA aminoacylation for protein translation
MG_037		NAD biosynthetic process
MG_038	glpK	glycerol-3-phosphate metabolic process, glycerol catabolic process, glycerol metabolic process
MG_041	ptsH	phosphoenolpyruvate-dependent sugar phosphotransferase system, regulation of transcription, DNA-templated, transcription, DNA-templated
MG_043	potB	transport
MG_044	potC	transport
MG_046	tsaD	tRNA threonylcarbamoyladenosine modification
MG_047	metK	one-carbon metabolic process, S-adenosylmethionine biosynthetic process
MG_048	ffh	SRP-dependent cotranslational protein targeting to membrane
MG_049	deoD	nucleoside metabolic process
MG_050	deoC	deoxyribonucleotide catabolic process, deoxyribose phosphate catabolic process
MG_051	deoA	pyrimidine nucleobase metabolic process, pyrimidine nucleoside metabolic process
MG_053	manB	carbohydrate metabolic process
MG_055		protein secretion
MG_058	prs	5-phosphoribose 1-diphosphate biosynthetic process, nucleoside metabolic process, nucleotide biosynthetic process
MG_062	fruA	phosphoenolpyruvate-dependent sugar phosphotransferase system
MG_066	tkt	metabolic process
MG_069	ptsG	phosphoenolpyruvate-dependent sugar phosphotransferase system
MG_070	rpsB	translation
MG_072	secA	protein import, protein targeting
MG_073	uvrB	nucleotide-excision repair, SOS response
MG_077	oppB	protein transport
MG_078	oppC	protein transport
MG_079	oppD	protein transport
MG_080	oppF	protein transport
MG_081	rplK	translation

Gene	Name	GO Biological Process Labels
MG_082	rplA	regulation of translation, translation
MG_084	tilS	tRNA processing
MG_085	hprK	carbohydrate metabolic process, regulation of carbohydrate metabolic process
MG_086	lgt	lipoprotein biosynthetic process, protein lipoylation
MG_087	rpsL	translation
MG_088	rpsG	translation
MG_090	rpsF	translation
MG_092	rpsR	translation
MG_093	rplI	translation
MG_094	dnaB	DNA replication, synthesis of RNA primer
MG_097	ung	base-excision repair
MG_098		regulation of translational fidelity
MG_099	gatA	translation
MG_100	gatB	translation
MG_101		transcription, DNA-templated
MG_102	trxB	removal of superoxide radicals
MG_106	def	translation
MG_110	rsgA	ribosome biogenesis
MG_111	pgi	gluconeogenesis, glycolytic process
MG_112	rpe	carbohydrate metabolic process
MG_113	asnS	asparaginyl-tRNA aminoacylation
MG_114	pgsA	phosphatidylglycerol biosynthetic process
MG_118	galE	galactose metabolic process
MG_119		carbohydrate transport
MG_120		transport
MG_121		transport
MG_122	topA	DNA topological change
MG_124	trxA	cell redox homeostasis, glycerol ether metabolic process
MG_126	trpS	tryptophanyl-tRNA aminoacylation
MG_128	nadK	NAD metabolic process, NADP biosynthetic process
MG_130	rny	mRNA catabolic process
MG_136	lysS	lysyl-tRNA aminoacylation
MG_139	rnj	rRNA processing
MG_141	nusA	DNA-templated transcription, termination, transcription antitermination
MG_143	rbfA	rRNA processing
MG_145	ribF	FAD biosynthetic process, FMN biosynthetic process, riboflavin biosynthetic process
MG_150	rpsJ	translation
MG_151	rplC	translation
MG_152	rplD	translation
MG_153	rplW	translation
MG_154	rplB	translation
MG_155	rpsS	translation
MG_156	rplV	translation
MG_157	rpsC	translation
MG_158	rplP	translation

Gene	Name	GO Biological Process Labels
MG_159	rpmC	translation
MG_160	rpsQ	translation
MG_161	rplN	translation
MG_162	rplX	translation
MG_163	rplE	translation
MG_164	rpsZ	translation
MG_165	rpsH	translation
MG_166	rplF	translation
MG_167	rplR	translation
MG_168	rpsE	translation
MG_169	rplO	translation
MG_170	secY	protein transport
MG_171	adk	AMP salvage
MG_174	rpmJ	translation
MG_175	rpsM	translation
MG_176	rpsK	translation
MG_177	rpoA	transcription, DNA-templated
MG_178	rplQ	translation
MG_179	ecfA1	transport
MG_180	ecfA2	transport
MG_182	truA	pseudouridine synthesis, tRNA processing
MG_188		transport
MG_189		transport
MG_191	mgpA	cytoadherence to microvasculature, mediated by symbiont protein
MG_192		cell adhesion
MG_194	pheS	phenylalanyl-tRNA aminoacylation
MG_195	pheT	phenylalanyl-tRNA aminoacylation
MG_197	rpmI	translation
MG_198	rplT	translation
MG_201	grpE	protein folding
MG_203	parE	DNA topological change
MG_204	parC	DNA topological change
MG_205	hrcA	regulation of transcription, DNA-templated, transcription, DNA-templated
MG_206	uvrC	DNA repair, SOS response
MG_208		tRNA threonylcarbamoyladenosine modification
MG_209		pseudouridine synthesis
MG_212	plsC	CDP-diacylglycerol biosynthetic process
MG_213	scpA	cell cycle, cell division, chromosome segregation
MG_214	scpB	cell division, chromosome separation
MG_215	pfkA	fructose 6-phosphate metabolic process
MG_218	hmw2	cytoadherence to microvasculature, mediated by symbiont protein, pathogenesis
MG_221	mraZ	transcription, DNA-templated
MG_224	ftsZ	cell cycle, cell division
MG_227	thyA	dTMP biosynthetic process, dTTP biosynthetic process
MG_228	folA	glycine biosynthetic process, nucleotide biosynthetic process, one-carbon metabolic process

Gene	Name	GO Biological Process Labels
MG_229	nrdF	deoxyribonucleotide biosynthetic process, DNA replication
MG_231	nrdE	DNA replication
MG_232	rplU	translation
MG_234	rpmA	translation
MG_235	nfo	DNA repair
MG_238	tig	cell cycle, cell division, protein folding
MG_239	lon	protein catabolic process
MG_240		biosynthetic process
MG_244	uvrD	DNA repair, DNA replication
MG_249	sigA	DNA-templated transcription, initiation
MG_251	glyQS	glycyl-tRNA aminoacylation
MG_252		RNA processing
MG_253	cysS	cysteinyl-tRNA aminoacylation
MG_254	ligA	DNA repair, DNA replication
MG_257	rpmE	translation
MG_261	dnaE	DNA replication
MG_264	coaE	coenzyme A biosynthetic process
MG_266	leuS	leucyl-tRNA aminoacylation
MG_270	lplA	protein lipoylation
MG_271	pdhD	cell redox homeostasis, glycolytic process
MG_272	pdhC	glycolytic process
MG_273	pdhB	glycolytic process
MG_274	pdhA	glycolytic process
MG_275	nox	cell redox homeostasis
MG_276	apt	adenine salvage, AMP salvage, purine ribonucleoside salvage
MG_278	spoT	guanosine tetraphosphate biosynthetic process
MG_282	greA	regulation of DNA-templated transcription, elongation, transcription, DNA-templated
MG_283	proS	prolyl-tRNA aminoacylation
MG_287		fatty acid biosynthetic process
MG_289	p37	transport
MG_291	p69	transport
MG_292	alaS	alanyl-tRNA aminoacylation
MG_293		lipid metabolic process
MG_295	mnmA	tRNA processing
MG_297	ftsY	SRP-dependent cotranslational protein targeting to membrane
MG_298	smc	chromosome condensation, sister chromatid cohesion
MG_299	pta	acetyl-CoA biosynthetic process
MG_300	pgk	glycolytic process
MG_301	gapA	adhesion of symbiont to host cell, glucose metabolic process, glycolytic process
MG_305	dnaK	protein folding
MG_311	rpsD	translation
MG_312	hmw1	cytoadherence to microvasculature, mediated by symbiont protein, pathogenesis
MG_315		DNA replication
MG_317	hmw3	cytoadherence to microvasculature, mediated by symbiont protein, pathogenesis
MG_318		cytoadherence, heterophilic cell-cell adhesion via plasma membrane cell adhesion molecules, pathogenesis

Gene	Name	GO Biological Process Labels
MG_323		potassium ion transport
MG_325	rpmG1	translation
MG_329	der	ribosome biogenesis
MG_334	valS	valyl-tRNA aminoacylation
MG_335	engB	cell cycle, cell division
MG_339	recA	DNA recombination, DNA repair, SOS response
MG_340	rpoC	transcription, DNA-templated
MG_341	rpoB	transcription, DNA-templated
MG_345	ileS	isoleucyl-tRNA aminoacylation
MG_346		tRNA processing
MG_351	ppa	phosphate-containing compound metabolic process
MG_352	recU	DNA recombination, DNA repair
MG_357	ackA	acetyl-CoA biosynthetic process, organic acid metabolic process
MG_358	ruvA	DNA recombination, DNA repair, SOS response
MG_359	ruvB	DNA recombination, DNA repair, SOS response
MG_361	rplJ	ribosome biogenesis, translation
MG_362	rplL	translation
MG_363	rpmF	translation
MG_367	rnc	mRNA processing, rRNA catabolic process, rRNA processing
MG_368	plsX	fatty acid biosynthetic process, phospholipid biosynthetic process
MG_369		glycerol metabolic process
MG_370		pseudouridine synthesis
MG_372	thiI	thiamine biosynthetic process, thiamine diphosphate biosynthetic process, tRNA thio-modification
MG_375	thrS	threonyl-tRNA aminoacylation
MG_378	argS	arginyl-tRNA aminoacylation
MG_379	mnmG	tRNA wobble uridine modification
MG_382	udk	CTP salvage, UMP salvage
MG_383	nadE	NAD biosynthetic process
MG_385		lipid metabolic process
MG_386		cytoadherence to microvasculature, mediated by symbiont protein
MG_387	era	ribosome biogenesis
MG_392	groL	protein refolding
MG_393	groS	protein folding
MG_394	glyA	glycine biosynthetic process, tetrahydrofolate interconversion
MG_396	rpiB	carbohydrate metabolic process, pentose-phosphate shunt, non-oxidative branch
MG_398	atpC	ATP synthesis coupled proton transport
MG_399	atpD	ATP synthesis coupled proton transport
MG_400	atpG	ATP synthesis coupled proton transport
MG_401	atpA	ATP synthesis coupled proton transport
MG_402	atpH	ATP synthesis coupled proton transport
MG_403	atpF	ATP synthesis coupled proton transport
MG_404	atpE	ATP hydrolysis coupled proton transport, ATP synthesis coupled proton transport
MG_405	atpB	ATP synthesis coupled proton transport
MG_407	eno	glycolytic process
MG_409		cellular phosphate ion homeostasis, negative regulation of phosphate metabolic process

Gene	Name	GO Biological Process Labels
MG_411	pstA	phosphate ion transmembrane transport
MG_417	rpsI	translation
MG_418	rplM	translation
MG_419	dnaX	DNA replication
MG_421	uvrA	nucleotide-excision repair, SOS response
MG_424	rpsO	translation
MG_426	rpmB	translation
MG_428		DNA-templated transcription, initiation
MG_429	ptsI	phosphoenolpyruvate-dependent sugar phosphotransferase system
MG_430	gpmI	glucose catabolic process, glycolytic process
MG_431	tpiA	gluconeogenesis, glycolytic process, pentose-phosphate shunt
MG_434	pyrH	'de novo' CTP biosynthetic process
MG_435	frr	translation
MG_437	cdsA	CDP-diacylglycerol biosynthetic process
MG_438		DNA restriction-modification system
MG_442	rbgA	ribosome biogenesis
MG_444	rplS	translation
MG_446	rpsP	translation
MG_448	msrB	protein repair, response to oxidative stress
MG_453	galU	biosynthetic process, UDP-glucose metabolic process
MG_454		response to oxidative stress
MG_455	tyrS	tyrosyl-tRNA aminoacylation
MG_458	hpt	IMP salvage, purine ribonucleoside salvage
MG_460	ldh	carbohydrate metabolic process, carboxylic acid metabolic process
MG_462	gltX	glutamyl-tRNA aminoacylation
MG_464	yidC	protein insertion into membrane, protein transport
MG_465	rnpA	tRNA processing
MG_466	rpmH	translation
MG_469	dnaA	DNA replication initiation, regulation of DNA replication
MG_473	rpmG2	translation
MG_476	secG	protein secretion
MG_481	rpsU	translation
MG_482	acpS	fatty acid biosynthetic process
MG_498	mutM	base-excision repair, nucleotide-excision repair
MG_517		enterobacterial common antigen biosynthetic process, glycerol metabolic process, membrane lipid biosynthetic process
MG_522	rpsT	translation

Table 11. 259 modelled *M.genitalium* genes with GO (Biological Process) terms. A published version of the table is available [56].

Unaffected GO Term Categories	Baseline GO N	Minesweeper_256 GO N
translation	56	56
glycolytic process	11	11
cell redox homeostasis	3	3
acetyl-CoA biosynthetic process	2	2
AMP salvage	2	2
CDP-diacylglycerol biosynthetic process	2	2
gluconeogenesis	2	2
glycine biosynthetic process	2	2
NAD biosynthetic process	2	2
nucleoside metabolic process	2	2
nucleotide biosynthetic process	2	2
one-carbon metabolic process	2	2
phenylalanyl-tRNA aminoacylation	2	2
purine ribonucleoside salvage	2	2
5-phosphoribose 1-diphosphate biosynthetic process	1	1
adenine salvage	1	1
adhesion of symbiont to host cell	1	1
alanyl-tRNA aminoacylation	1	1
arginyl-tRNA aminoacylation	1	1
asparaginyl-tRNA aminoacylation	1	1
CTP salvage	1	1
cysteinyl-tRNA aminoacylation	1	1
de novo' CTP biosynthetic process	1	1
deoxyribonucleotide biosynthetic process	1	1
DNA biosynthetic process	1	1
DNA replication synthesis of RNA primer	1	1
DNA replication initiation	1	1
DNA-templated transcription termination	1	1
dTDP biosynthetic process	1	1
enterobacterial common antigen biosynthetic process	1	1
FAD biosynthetic process	1	1
FMN biosynthetic process	1	1
fructose 1 6-bisphosphate metabolic process	1	1
fructose 6-phosphate metabolic process	1	1
galactose metabolic process	1	1
glucose catabolic process	1	1
glucose metabolic process	1	1
glutamyl-tRNA aminoacylation	1	1
glycerol catabolic process	1	1
glycerol ether metabolic process	1	1
glycerol-3-phosphate metabolic process	1	1
glycyl-tRNA aminoacylation	1	1
guanosine tetraphosphate biosynthetic process	1	1
histidine biosynthetic process	1	1
histidyl-tRNA aminoacylation	1	1

Unaffected GO Term Categories	Baseline GO N	Minesweeper_256 GO N
IMP salvage	1	1
isoleucyl-tRNA aminoacylation	1	1
leucyl-tRNA aminoacylation	1	1
lysyl-tRNA aminoacylation	1	1
membrane lipid biosynthetic process	1	1
metabolic process	1	1
methionine biosynthetic process	1	1
methionyl-tRNA aminoacylation	1	1
mRNA processing	1	1
NAD metabolic process	1	1
NADP biosynthetic process	1	1
organic acid metabolic process	1	1
pentose-phosphate shunt	1	1
pentose-phosphate shunt non-oxidative branch	1	1
phosphate-containing compound metabolic process	1	1
phosphatidylglycerol biosynthetic process	1	1
phospholipid biosynthetic process	1	1
potassium ion transport	1	1
prolyl-tRNA aminoacylation	1	1
purine nucleotide biosynthetic process	1	1
pyrimidine nucleobase metabolic process	1	1
pyrimidine nucleoside metabolic process	1	1
regulation of DNA replication	1	1
regulation of DNA-templated transcription elongation	1	1
regulation of translation	1	1
regulation of translational fidelity	1	1
removal of superoxide radicals	1	1
response to heat	1	1
riboflavin biosynthetic process	1	1
rRNA catabolic process	1	1
S-adenosylmethionine biosynthetic process	1	1
selenocysteinyl-tRNA(Sec) biosynthetic process	1	1
seryl-tRNA aminoacylation	1	1
tetrahydrofolate interconversion	1	1
thiamine biosynthetic process	1	1
thiamine diphosphate biosynthetic process	1	1
threonyl-tRNA aminoacylation	1	1
transcription antitermination	1	1
tRNA aminoacylation for protein translation	1	1
tRNA modification	1	1
tRNA thio-modification	1	1
tRNA wobble uridine modification	1	1
tryptophanyl-tRNA aminoacylation	1	1
tyrosyl-tRNA aminoacylation	1	1
UDP-glucose metabolic process	1	1

Unaffected GO Term Categories	Baseline GO N	Minesweeper_256 GO N
valyl-tRNA aminoacylation	1	1
Reduced GO Term Categories	Baseline GO N	Minesweeper_256 GO N
DNA replication	10	9
protein folding	5	4
tRNA processing	5	4
DNA topological change	5	4
Transcription DNA-templated	9	7
phosphoenolpyruvate-dependent sugar phosphotransferase system	4	3
protein transport	6	4
fatty acid biosynthetic process	3	2
regulation of transcription DNA-templated	3	2
transport	10	6
carbohydrate metabolic process	5	3
cell cycle	4	2
glycerol metabolic process	4	2
biosynthetic process	2	1
DNA-templated transcription initiation	2	1
protein lipoylation	2	1
UMP salvage	2	1
cell division	5	2
ribosome biogenesis	5	2
pseudouridine synthesis	3	1
rRNA processing	3	1
DNA repair	8	1
Removed GO Term Categories	Baseline GO N	Minesweeper_256 GO N
ATP synthesis coupled proton transport	8	0
cytoadherence to microvasculature mediated by symbiont protein	6	0
SOS response	6	0
DNA recombination	4	0
pathogenesis	4	0
nucleotide-excision repair	3	0
base-excision repair	2	0
lipid metabolic process	2	0
protein secretion	2	0
response to oxidative stress	2	0
SRP-dependent cotranslational protein targeting to membrane	2	0
tRNA threonylcarbamoyladenosine modification	2	0
ATP hydrolysis coupled proton transport	1	0
carbohydrate transport	1	0
carboxylic acid metabolic process	1	0
cell adhesion	1	0
cellular phosphate ion homeostasis	1	0
cellular protein modification process	1	0
chromosome condensation	1	0
chromosome segregation	1	0

Removed GO Term Categories	Baseline GO N	Minesweeper_256 GO N
chromosome separation	1	0
coenzyme A biosynthetic process	1	0
deoxyribonucleotide catabolic process	1	0
deoxyribose phosphate catabolic process	1	0
DNA restriction-modification system	1	0
dTMP biosynthetic process	1	0
dTTP biosynthetic process	1	0
heterophilic cell-cell adhesion via plasma membrane cell adhesion molecules	1	0
lipoprotein biosynthetic process	1	0
mRNA catabolic process	1	0
negative regulation of phosphate metabolic process	1	0
phosphate ion transmembrane transport	1	0
protein catabolic process	1	0
protein import	1	0
protein insertion into membrane	1	0
protein refolding	1	0
protein repair	1	0
protein targeting	1	0
regulation of carbohydrate metabolic process	1	0
sister chromatid cohesion	1	0
RNA processing	1	0
uracil salvage	1	0

Table 12. Minesweeper_256 gene deletions impact on GO terms. A published version of the table is available [56].

I conducted further analysis, as some of these removals could be of concern to the longevity of the *in-silico* cell. The GO term proton transport applies to the genes MG_398-405, which form ATP synthase, an enzyme that generates ATP using energy from protons transferring across the cell membrane. This removes one pathway for producing ATP, but the minimal genome still contains intact phosphoglycerate kinase (MG_300) and pyruvate kinase (MG_216) that both produce ATP as part of glycolysis. Additionally, there are 13 reversible reactions that produce ATP in the reverse reaction (Table 13).

Enzyme	Reaction	Genes
ATP synthase (four protons for one ATP) (periplasm)	ADP[c] + (4) H[e] + H[c] + PI[c] <==> ATP[c] + (4) H[c] + H2O[c]	MG_398_399_400_401_402_403_404_405_22MER
phosphoglycerate kinase	[c]: ADP + DPG <==> ATP + G3P	MG_300_MONOMER
pyruvate kinase	[c]: ADP + H + PEP <==> ATP + PYR	MG_216_TETRAMER
acetate kinase	[c]: ACTP + ADP <==> AC + ATP	MG_357_DIMER
deoxyadenylate kinase (dADP)	[c]: ATP + DAMP <==> ADP + DADP	MG_171_MONOMER
choline kinase	[c]: ATP + CHOL ==> ADP + H + pCHOL	MG_356_MONOMER
cytidylate kinase (dCMP)	[c]: ATP + DCMP <==> ADP + DCDP	MG_330_MONOMER
cytidylate kinase (CMP)	[c]: ATP + CMP <==> ADP + CDP	MG_330_MONOMER
UMP kinase	[c]: ATP + UMP <==> ADP + UDP	MG_434_HEXAMER
methylenetetrahydrofolate cyclohydrase	[c]: ATP + FTHF10 + (3) H <==> ADP + METTHF + PI	MG_245_MONOMER
guanylate kinase (GMP:ATP)	[c]: ATP + GMP <==> ADP + GDP	MG_107_DIMER
deoxyguanylate kinase (dGMP:ATP)	[c]: ATP + DGMP <==> ADP + DGDP	MG_107_DIMER
NAD kinase	[c]: ATP + NAD <==> ADP + NADP	MG_128_HEXAMER
phosphoribosylpyrophosphate synthetase	[c]: ATP + R5P <==> AMP + H + PRPP	MG_058_HEXAMER
thiamine-phosphate kinase	[c]: ATP + THMP <==> ADP + TPP	MG_372_DIMER
thiamine kinase	[c]: ATP + DTMP <==> ADP + DTDP	MG_006_DIMER

Table 13. ATP producing reactions in the *M.genitalium* whole-cell model. Data from Karr *et al.* Supplementary Table mmc4-O [6]. A published version of the table is available [56].

The GO term DNA recombination applies to the genes MG_339, MG_352, MG_358, MG_359, which conduct half of the steps in homologous recombination double strand break repair. This process, as well as nucleotide excision repair and base excision repair are removed from the cell. However, direct damage reversal (MG_254) and DNA polymerase (MG_001) are still present. *DisA* (MG_105), the DNA damage sensor, has currently been deleted but I believe this is due the single-generation nature of the *M.genitalium* whole-cell model; the cell can survive without successful DNA repair for a single generation *in-silico*, whereas it cannot for multiple generations *in-vivo*; evidenced by Karr *et al.* Supplementary Table S2G [6] labelling the single knockout as a false non-essential.

DisA is the only dedicated process for recognising DNA damage in *M.genitalium*, and is believed to signal lack of damage by producing a secondary messenger molecule, which it stops producing when it binds to damaged DNA (Karr *et al.* Supplementary mmc1, pg 40)) [6]. Most knowledge in how it functions comes from other species (*B.subtilis*), but *M.genitalium* appears to lack the rest of the signalling system present in

other species. This lack of knowledge simplifies what can be implemented in the model. The DNA damage sub-model models random and radiation-induced DNA damage, while the DNA repair sub-model models the binding of free DisA to DNA lesions (Karr *et al.* Supplementary mmc1, pg 42)) [6], depending on enzyme availability. For the actual DNA repair functions to occur, the model conducts a series of general steps:

1. Identify chromosomes sites that are substrates for the repair system.
    a. i.e. areas of DNA damage bound by *DisA*
2. Eliminate sites that neighbor damage, or which are occupied by proteins (other than the DNA damage scanning protein *DisA*).
3. Compute maximum number of repair reactions that can occur based on metabolite and enzyme availability, and kinetics.
4. Randomly select among eligible repair sites for repair.
5. Update state of DNA and update metabolites.
6. Free any *DisA* that was previously bound to the repaired site.

It appears there is little to be altered or updated in the model to better capture the biological function of *DisA* until more is known of how it functions in *M.genitalium in-vivo*. The incorrect essentiality classification from its single gene knockout can, I believe, only be solved by implementing multi-generational simulations in the *M.genitalium* whole-cell model.

The GO term chromosome segregation applies only to MG_213, where it is listed as its tertiary function. The genes in the model that actually conduct chromosome segregation (MG_470, MG_221, MG_387, MG_384, MG_203, MG_204, MG_224 (Karr *et al.* Supplementary mmc1, pg 34)) [6] are all present in the minimal genome, but do not have an associated GO term. This underlines the use of caution when using GO terms and the need for secondary analysis.

The gene deletions in Minesweeper_256 reduce the ability of the *in-silico* cell to interact with the environment and defend against external forces. They also cause a reduction in control, from transport to regulation to genome management, and prune metabolic processes and metabolites. This leaves the *in-silico* cell alive, but more vulnerable to

external and internal pressures, less capable of responding to change, and more reliant on internal processes occurring by chance.

In comparison, GAMA_237 has 163 genes (69% coverage) with GO terms on UniProt [160], with 73 genes with no GO terms. The 165 genes deleted reduced 18 (11%) GO categories and removed 54 (35%) GO categories, 38 of which were associated with a single gene (Table 14). The gene deletions unique to GAMA_237 can be seen in Table 15, 17, and 18.

One reduced GO category was less affected compared to Minesweeper_256 (glycerol metabolic process) and one unaffected GO category was unique to GAMA_237 (phosphate ion transmembrane transport). Three GO categories were reduced further in GAMA_237: DNA transcription, DNA transcription regulation, and transport (ABC transporters) (Table 18).

Categories that were removed solely in GAMA_237 include: DNA transcription (termination, regulation, elongation, antitermination, initiation); tRNA (processing, modification); and rRNA catabolic process.

Unaffected GO Term Categories	Baseline GO N	GAMA_237 GO N
translation	56	56
glycolytic process	11	11
cell redox homeostasis	3	3
acetyl-CoA biosynthetic process	2	2
AMP salvage	2	2
CDP-diacylglycerol biosynthetic process	2	2
gluconeogenesis	2	2
glycine biosynthetic process	2	2
NAD biosynthetic process	2	2
nucleoside metabolic process	2	2
nucleotide biosynthetic process	2	2
one-carbon metabolic process	2	2
phenylalanyl-tRNA aminoacylation	2	2
purine ribonucleoside salvage	2	2
5-phosphoribose 1-diphosphate biosynthetic process	1	1
adenine salvage	1	1
adhesion of symbiont to host cell	1	1
alanyl-tRNA aminoacylation	1	1
arginyl-tRNA aminoacylation	1	1
asparaginyl-tRNA aminoacylation	1	1
CTP salvage	1	1
cysteinyl-tRNA aminoacylation	1	1
de novo' CTP biosynthetic process	1	1
deoxyribonucleotide biosynthetic process	1	1
DNA biosynthetic process	1	1
DNA replication synthesis of RNA primer	1	1
DNA replication initiation	1	1
dTDP biosynthetic process	1	1
enterobacterial common antigen biosynthetic process	1	1
FAD biosynthetic process	1	1
FMN biosynthetic process	1	1
fructose 1 6-bisphosphate metabolic process	1	1
fructose 6-phosphate metabolic process	1	1
galactose metabolic process	1	1
glucose catabolic process	1	1
glucose metabolic process	1	1
glutamyl-tRNA aminoacylation	1	1
glycerol catabolic process	1	1
glycerol ether metabolic process	1	1
glycerol-3-phosphate metabolic process	1	1
glycyl-tRNA aminoacylation	1	1
guanosine tetraphosphate biosynthetic process	1	1
histidine biosynthetic process	1	1
histidyl-tRNA aminoacylation	1	1

Unaffected GO Term Categories	Baseline GO N	GAMA_237 GO N
IMP salvage	1	1
isoleucyl-tRNA aminoacylation	1	1
leucyl-tRNA aminoacylation	1	1
lysyl-tRNA aminoacylation	1	1
membrane lipid biosynthetic process	1	1
metabolic process	1	1
methionine biosynthetic process	1	1
methionyl-tRNA aminoacylation	1	1
NAD metabolic process	1	1
NADP biosynthetic process	1	1
organic acid metabolic process	1	1
pentose-phosphate shunt	1	1
pentose-phosphate shunt non-oxidative branch	1	1
phosphate ion transmembrane transport	1	1
phosphate-containing compound metabolic process	1	1
phosphatidylglycerol biosynthetic process	1	1
phospholipid biosynthetic process	1	1
potassium ion transport	1	1
prolyl-tRNA aminoacylation	1	1
purine nucleotide biosynthetic process	1	1
pyrimidine nucleobase metabolic process	1	1
pyrimidine nucleoside metabolic process	1	1
regulation of DNA replication	1	1
regulation of translation	1	1
regulation of translational fidelity	1	1
removal of superoxide radicals	1	1
response to heat	1	1
riboflavin biosynthetic process	1	1
S-adenosylmethionine biosynthetic process	1	1
selenocysteinyl-tRNA(Sec) biosynthetic process	1	1
seryl-tRNA aminoacylation	1	1
tetrahydrofolate interconversion	1	1
threonyl-tRNA aminoacylation	1	1
tRNA aminoacylation for protein translation	1	1
tRNA modification	1	1
tryptophanyl-tRNA aminoacylation	1	1
tyrosyl-tRNA aminoacylation	1	1
UDP-glucose metabolic process	1	1
valyl-tRNA aminoacylation	1	1
Reduced GO Term Categories	Baseline GO N	GAMA_237 GO N
DNA replication	10	9
DNA topological change	5	4
protein folding	5	4
glycerol metabolic process	4	3
phosphoenolpyruvate-dependent sugar phosphotransferase system	4	3

Reduced GO Term Categories	Baseline GO N	GAMA_237 GO N
protein transport	6	4
protein lipoylation	2	1
fatty acid biosynthetic process	3	2
carbohydrate metabolic process	5	3
cell cycle	4	2
biosynthetic process	2	1
UMP salvage	2	1
transport	10	4
cell division	5	2
ribosome biogenesis	5	2
regulation of transcription DNA-templated	3	1
transcription DNA-templated	9	2
DNA repair	8	1
Removed GO Term Categories	Baseline GO N	GAMA_237 GO N
ATP synthesis coupled proton transport	8	0
cytoadherence to microvasculature mediated by symbiont protein	6	0
SOS response	6	0
tRNA processing	5	0
DNA recombination	4	0
pathogenesis	4	0
nucleotide-excision repair	3	0
pseudouridine synthesis	3	0
rRNA processing	3	0
base-excision repair	2	0
DNA-templated transcription initiation	2	0
lipid metabolic process	2	0
protein secretion	2	0
response to oxidative stress	2	0
SRP-dependent cotranslational protein targeting to membrane	2	0
tRNA threonylcarbamoyladenosine modification	2	0
ATP hydrolysis coupled proton transport	1	0
carbohydrate transport	1	0
carboxylic acid metabolic process	1	0
cell adhesion	1	0
cellular phosphate ion homeostasis	1	0
cellular protein modification process	1	0
chromosome condensation	1	0
chromosome segregation	1	0
chromosome separation	1	0
coenzyme A biosynthetic process	1	0
deoxyribonucleotide catabolic process	1	0
deoxyribose phosphate catabolic process	1	0
DNA restriction-modification system	1	0
DNA-templated transcription termination	1	0
dTMP biosynthetic process	1	0

Removed GO Term Categories	Baseline GO N	GAMA_237 GO N
dTTP biosynthetic process	1	0
heterophilic cell-cell adhesion via plasma membrane cell adhesion molecules	1	0
lipoprotein biosynthetic process	1	0
mRNA catabolic process	1	0
mRNA processing	1	0
negative regulation of phosphate metabolic process	1	0
protein catabolic process	1	0
protein import	1	0
protein insertion into membrane	1	0
protein refolding	1	0
protein repair	1	0
protein targeting	1	0
regulation of carbohydrate metabolic process	1	0
regulation of DNA-templated transcription elongation	1	0
RNA processing	1	0
rRNA catabolic process	1	0
sister chromatid cohesion	1	0
thiamine biosynthetic process	1	0
thiamine diphosphate biosynthetic process	1	0
transcription antitermination	1	0
tRNA thio-modification	1	0
tRNA wobble uridine modification	1	0
uracil salvage	1	0

Table 14. GAMA_237 gene deletions impact on GO terms. A published version of the table is available [56].

The GO analysis of GAMA_237, when compared to that of Minesweeper_256, suggests a further reduction of both internal control and reactivity to the external environment. These reductions are discussed further below.

4.4.7 Low Essential Genes

I analysed Minesweeper_256 and GAMA_237 to determine whether these were different minimal genomes or GAMA_237 was an extension of Minesweeper_256. I compared unmodified *M.genitalium*, Minesweeper_256, and GAMA_237 genomes (Figure 15), which highlighted 141 shared deletions and gene deletions unique to each minimal genome (Table 15).

	Shared Deletions	Unique to GAMA_237	Unique to Minesweeper_256
1	MG_009	MG_008	MG_033
2	MG_012	MG_022	MG_410
3	MG_014	MG_039	MG_411
4	MG_015	MG_084	MG_412
5	MG_020	MG_104	
6	MG_027	MG_141	
7	MG_029	MG_177	
8	MG_030	MG_182	
9	MG_040	MG_249	
10	MG_046	MG_282	
11	MG_048	MG_289	
12	MG_050	MG_290	
13	MG_052	MG_291	
14	MG_055	MG_295	
15	MG_059	MG_340	
16	MG_061	MG_341	
17	MG_062	MG_347	
18	MG_063	MG_367	
19	MG_064	MG_372	
20	MG_065	MG_379	
21	MG_072	MG_427	
22	MG_073	MG_445	
23	MG_075	MG_465	
24	MG_083		
25	MG_085		
26	MG_086		
27	MG_097		
28	MG_101		
29	MG_476		
30	MG_105		
31	MG_109		
32	MG_110		
33	MG_119		
34	MG_120		
35	MG_121		
36	MG_122		
37	MG_123		
38	MG_127		
39	MG_130		
40	MG_132		
41	MG_139		
42	MG_143		
43	MG_149		

	Shared Deletions		Shared Deletions		Shared Deletions
44	MG_170	88	MG_316	132	MG_447
45	MG_172	89	MG_317	133	MG_448
46	MG_183	90	MG_318	134	MG_454
47	MG_184	91	MG_324	135	MG_457
48	MG_186	92	MG_327	136	MG_460
49	MG_187	93	MG_329	137	MG_463
50	MG_188	94	MG_333	138	MG_464
51	MG_189	95	MG_335	139	MG_467
52	MG_190	96	MG_336	140	MG_468
53	MG_191	97	MG_339	141	MG_526
54	MG_192	98	MG_344		
55	MG_200	99	MG_346		
56	MG_205	100	MG_349		
57	MG_206	101	MG_352		
58	MG_208	102	MG_353		
59	MG_209	103	MG_355		
60	MG_210	104	MG_356		
61	MG_482	105	MG_358		
62	MG_213	106	MG_359		
63	MG_214	107	MG_369		
64	MG_217	108	MG_370		
65	MG_218	109	MG_376		
66	MG_225	110	MG_380		
67	MG_226	111	MG_385		
68	MG_227	112	MG_386		
69	MG_235	113	MG_390		
70	MG_236	114	MG_391		
71	MG_239	115	MG_392		
72	MG_240	116	MG_393		
73	MG_244	117	MG_398		
74	MG_252	118	MG_399		
75	MG_259	119	MG_400		
76	MG_262	120	MG_401		
77	MG_498	121	MG_402		
78	MG_264	122	MG_403		
79	MG_265	123	MG_404		
80	MG_277	124	MG_405		
81	MG_288	125	MG_408		
82	MG_293	126	MG_409		
83	MG_297	127	MG_421		
84	MG_298	128	MG_425		
85	MG_309	129	MG_428		
86	MG_310	130	MG_438		
87	MG_312	131	MG_442		

Table 15. Shared and unique gene deletions of Minesweeper_256 and GAMA_237. A published version of the table is available [56], as is the simulation data [161].

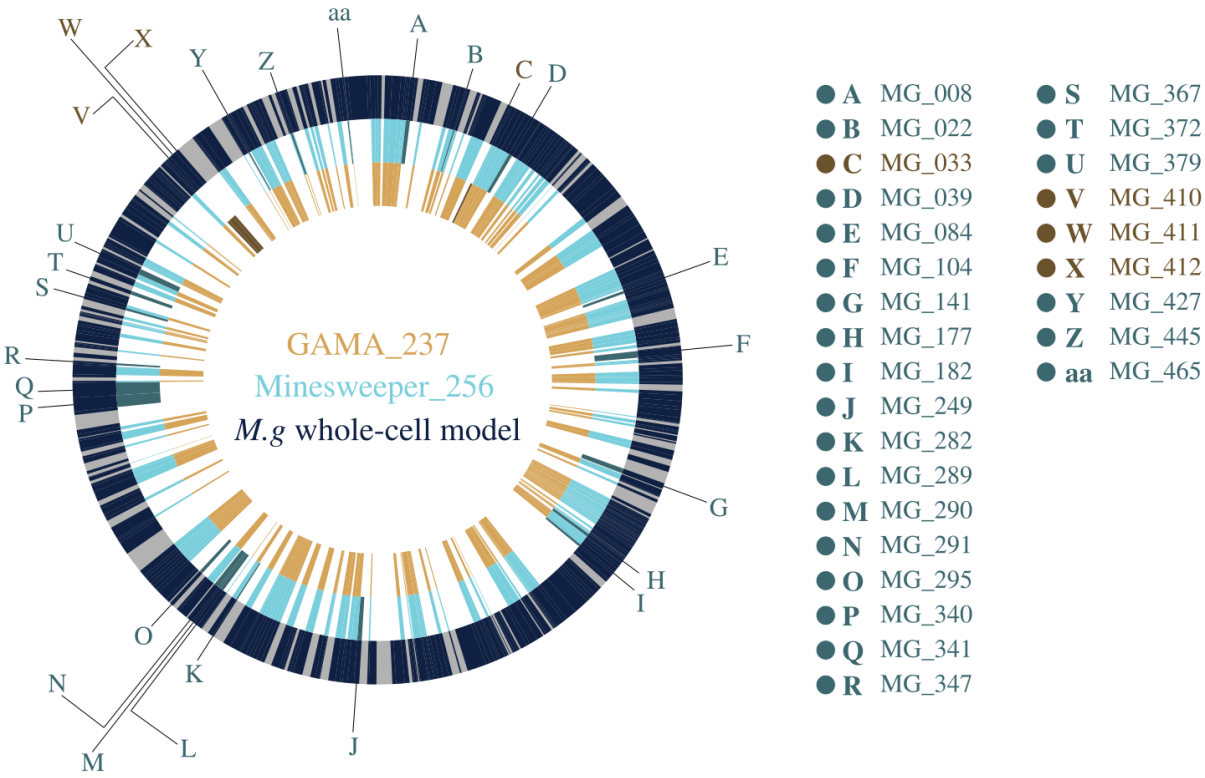

Figure 15. Genome comparison of the whole-cell model, Minesweeper_256, and GAMA_237. *This figure was created by co-author Sophie Landon for our paper [56] using data produced by myself (Minesweeper_256) and Oliver Chalkley (GAMA_237).* The outer ring displays the *M.genitalium* genome (525 genes in total), with modelled genes (401) in navy and unmodelled genes (124, with unknown function) in grey. The middle ring displays the reduced Minesweeper_256 (256 genes) genome in light blue, with genes present in Minesweeper_265 but not in GAMA_237 in dark blue. The inner ring displays the reduced GAMA_237 (237 genes) genome in light yellow, with genes present in GAMA_237 but not in Minesweeper_265 in dark yellow. Figure produced from published *M.genitalium* genetic data [6,58], with genetic data for Minesweeper_256 and GAMA_237 available in Table 15.

Our genome comparison found that Minesweeper_256 removed four genes, and GAMA_237 removed five genes (Table 16), that could not be removed from the other genome (either individually or as a group) without preventing cellular division. An additional gene, MG_305, could not be removed in either GAMA_237 or Minesweeper_256. I confirmed that these ten genes were individually non-essential (Table 6) and that nine of the genes have low essentiality [10]. To identify the cause of this synthetic lethality I attempted to match the functions of the low essential genes together, anticipating redundant essential gene pairs or groups.

Gene	Annotation	Function	Removed In	Present In
MG_039	Uncharacterised	Probable catalyst of redox reactions.	GAMA_237	Minesweeper_256
MG_289	*p37*	High-affinity transport system protein attached to cell membrane.	GAMA_237	Minesweeper_256
MG_290	*p29*	Probable ATP-binding cassette (ABC) transporter.	GAMA_237	Minesweeper_256
MG_291	*p69*	Permease (ABC membrane transporter) protein.	GAMA_237	Minesweeper_256
MG_427	Unnamed	Reduces peroxides, protecting against oxidative stress.	GAMA_237	Minesweeper_256
MG_033	*glpF*	Facilitates glycerol across the membrane.	Minesweeper_256	GAMA_237
MG_410	*pstB*	Imports phosphate (part of PstSACB ABC complex).	Minesweeper_256	GAMA_237
MG_411	*pstA*	Permease protein for phosphate transport system.	Minesweeper_256	GAMA_237
MG_412	Uncharacterised	Probable phosphate ion binding attached to cell membrane.	Minesweeper_256	GAMA_237
MG_305	*dnaK*	Chaperone protein involved in refolding mis/unfolded heat shock proteins.	*M.g** whole-cell model	GAMA_237 and Minesweeper_256

Table 16. Low essential genes from Minesweeper_256 and GAMA_237 genomic contexts. Protein annotation and function obtained from UniProt [160], based on Fraser *et al.*'s **M.genitalium* G37 genome [58].

I found two genes in GAMA_237 (MG_289, MG_291) that had matching GO terms with the gene MG_411 in Minesweeper_256. These, and three other adjacent genes on the genome, were tested by combinatorial gene knockouts in an unmodified *M.genitalium* whole-cell model genome (Table 17). MG_289, MG_290, MG_291 were found to form a functional group, as were MG_410, MG_411, MG_412. These genes could be deleted individually and in functional groups from an otherwise unmodified *M.genitalium* whole-cell genome and produce a dividing *in-silico* cell. However, any double gene deletion combination that involved one gene from each functional group resulted in a cell that could not produce RNA, produce protein, replicate DNA, grow or divide.

	Gene Knockouts	Outcome
1	MG_289,	Dividing
2	MG_289,	Dividing
3	MG_289,	Dividing
4	MG_290,	Dividing
5	MG_290,	Dividing
6	MG_290,	Dividing
7	MG_291,	Dividing
8	MG_291,	Dividing
9	MG_291,	Dividing
10	MG_410,	Dividing
11	MG_410,	Dividing
12	MG_410,	Dividing
13	MG_411,	Dividing
14	MG_411,	Dividing
15	MG_411,	Dividing
16	MG_412,	Dividing
17	MG_412,	Dividing
18	MG_412,	Dividing
19	MG_289, MG_291,	Dividing
20	MG_289, MG_291,	Dividing
21	MG_289, MG_291,	Dividing
22	MG_290, MG_291,	Dividing
23	MG_290, MG_291,	Dividing
24	MG_290, MG_291,	Dividing
25	MG_290, MG_289,	Dividing
26	MG_290, MG_289,	Dividing
27	MG_290, MG_289,	Dividing
28	MG_410, MG_411, MG_412,	Dividing
29	MG_410, MG_411, MG_412,	Dividing
30	MG_410, MG_411, MG_412,	Dividing
31	MG_289, MG_410,	No Division
32	MG_289, MG_410,	No Division
33	MG_289, MG_410,	No Division
34	MG_289, MG_411,	No Division
35	MG_289, MG_411,	No Division
36	MG_289, MG_411,	No Division
37	MG_289, MG_412,	No Division
38	MG_289, MG_412,	No Division
39	MG_289, MG_412,	No Division
40	MG_291, MG_410,	No Division
41	MG_291, MG_410,	No Division
42	MG_291, MG_410,	No Division
43	MG_291, MG_411,	No Division
44	MG_291, MG_411,	No Division

	Gene Knockouts	Outcome
45	MG_291, MG_411,	No Division
46	MG_291, MG_412,	No Division
47	MG_291, MG_412,	No Division
48	MG_291, MG_412,	No Division
49	MG_290, MG_410,	No Division
50	MG_290, MG_410,	No Division
51	MG_290, MG_410,	No Division
52	MG_290, MG_411,	No Division
53	MG_290, MG_411,	No Division
54	MG_290, MG_411,	No Division
55	MG_290, MG_412,	No Division
56	MG_290, MG_412,	No Division
57	MG_290, MG_412,	No Division
58	MG_289, MG_291, MG_410,	No Division
59	MG_289, MG_291, MG_410,	No Division
60	MG_289, MG_291, MG_410,	No Division
61	MG_289, MG_291, MG_411,	No Division
62	MG_289, MG_291, MG_411,	No Division
63	MG_289, MG_291, MG_411,	No Division
64	MG_289, MG_291, MG_412,	No Division
65	MG_289, MG_291, MG_412,	No Division
66	MG_289, MG_291, MG_412,	No Division
67	MG_290, MG_291, MG_410,	No Division
68	MG_290, MG_291, MG_410,	No Division
69	MG_290, MG_291, MG_410,	No Division
70	MG_290, MG_291, MG_411,	No Division
71	MG_290, MG_291, MG_411,	No Division
72	MG_290, MG_291, MG_411,	No Division
73	MG_290, MG_291, MG_412,	No Division
74	MG_290, MG_291, MG_412,	No Division
75	MG_290, MG_291, MG_412,	No Division
76	MG_289, MG_290, MG_410,	No Division
77	MG_289, MG_290, MG_410,	No Division
78	MG_289, MG_290, MG_410,	No Division
79	MG_289, MG_290, MG_411,	No Division
80	MG_289, MG_290, MG_411,	No Division
81	MG_289, MG_290, MG_411,	No Division
82	MG_289, MG_290, MG_412,	No Division
83	MG_289, MG_290, MG_412,	No Division
84	MG_289, MG_290, MG_412,	No Division
85	MG_410, MG_411, MG_289,	No Division
86	MG_410, MG_411, MG_289,	No Division
87	MG_410, MG_411, MG_289,	No Division
88	MG_410, MG_411, MG_291,	No Division
89	MG_410, MG_411, MG_291,	No Division

	Gene Knockouts	Outcome
90	MG_410, MG_411, MG_291,	No Division
91	MG_410, MG_412, MG_289,	No Division
92	MG_410, MG_412, MG_289,	No Division
93	MG_410, MG_412, MG_289,	No Division
94	MG_410, MG_412, MG_291,	No Division
95	MG_410, MG_412, MG_291,	No Division
96	MG_410, MG_412, MG_291,	No Division
97	MG_411, MG_412, MG_289,	No Division
98	MG_411, MG_412, MG_289,	No Division
99	MG_411, MG_412, MG_289,	No Division
100	MG_411, MG_412, MG_291,	No Division
101	MG_411, MG_412, MG_291,	No Division
102	MG_411, MG_412, MG_291,	No Division
103	MG_410, MG_411, MG_290,	No Division
104	MG_410, MG_411, MG_290,	No Division
105	MG_410, MG_411, MG_290,	No Division
106	MG_410, MG_412, MG_290,	No Division
107	MG_410, MG_412, MG_290,	No Division
108	MG_410, MG_412, MG_290,	No Division
109	MG_411, MG_412, MG_290,	No Division
110	MG_411, MG_412, MG_290,	No Division
111	MG_411, MG_412, MG_290,	No Division
112	MG_410, MG_411, MG_412, MG_289,	No Division
113	MG_410, MG_411, MG_412, MG_289,	No Division
114	MG_410, MG_411, MG_412, MG_289,	No Division
115	MG_410, MG_411, MG_412, MG_290,	No Division
116	MG_410, MG_411, MG_412, MG_290,	No Division
117	MG_410, MG_411, MG_412, MG_290,	No Division
118	MG_410, MG_411, MG_412, MG_291,	No Division
119	MG_410, MG_411, MG_412, MG_291,	No Division
120	MG_410, MG_411, MG_412, MG_291,	No Division
121	MG_410, MG_411, MG_412, MG_289, MG_291,	No Division
122	MG_410, MG_411, MG_412, MG_289, MG_291,	No Division
123	MG_410, MG_411, MG_412, MG_289, MG_291,	No Division
124	MG_410, MG_411, MG_412, MG_289, MG_290,	No Division
125	MG_410, MG_411, MG_412, MG_289, MG_290,	No Division
126	MG_410, MG_411, MG_412, MG_289, MG_290,	No Division
127	MG_410, MG_411, MG_412, MG_290, MG_291,	No Division
128	MG_410, MG_411, MG_412, MG_290, MG_291,	No Division
129	MG_410, MG_411, MG_412, MG_290, MG_291,	No Division
130	MG_410, MG_411, MG_412, MG_289, MG_290, MG_291,	No Division

Table 17. Testing potentially redundant functional groups using combinatorial gene knockouts. A published version of the table is available [56], as is the simulation data [161].

M.genitalium only has two external sources of phosphate, inorganic phosphate and phosphonate. MG_410, MG_411, and MG_412 transport inorganic phosphate into the cell, and MG_289, MG_290, and MG_291 transport phosphonate into the cell (Glass *et al.* Supplementary Table 6 [61]). These phosphate sources proved to be a key difference between the minimal genomes. Minesweeper_256 removed the phosphate transport genes, relying on phosphonate as the sole phosphate source. GAMA_237 removed the phosphonate transport genes, relying on inorganic phosphate as the sole phosphate source. This can be seen in the GO term analysis, the phosphate ion transmembrane transport is still present in GAMA_237 but not in Minesweeper_256 (Table 12 and 13).

It has previously been theorised that individual bacterial species will have multiple minimal genomes [167,168], with different gene content depending on the environment, and which evolutionarily redundant cellular pathways were selected during reduction. The results I reported above suggest that one of these selections is the sourcing of phosphate, with minimal genomes differing by choice of phosphate transport genes and associated processing stages, equivalent to the *phn* gene cluster in *Escherichia coli* [169]. However, I could not find any annotated phosphonate processing genes that had been removed in GAMA_237, following GAMA_237 removing the phosphonate transport genes. I suspect that further "pivot points" (the selection of one redundant cellular pathway over another during genome reduction) will be identified in future *in-vivo* and *in-silico* bacterial reductions, increasing the number of minimal genomes per bacterial species.

I additionally investigated MG_305 (*DnaK*), the gene that neither Minesweeper_256 or GAMA_237 could remove (Table 16). This shares the protein folding GO term with four other genes: MG_019, MG_201, MG_238, and MG_393. Three were unmodified in either genome, but MG_393 (*GroES*) was removed by both Minesweeper_256 and GAMA_237, indicating a potential redundant essential relationship. However, knocking in MG_393 and knocking out MG_305 still produces a non-dividing cell in both GAMA and Minesweeper (Table 18). At this stage, I thought that MG_305 had additional redundant pair relationships that have already suffered one deletion shared by both GAMA_237 and Minesweeper_256 (Table 15).

Further investigation identified MG_392 (*GroEL*). This is a protein chaperone, which similar to DnaK is believed to assist in late protein folding (Karr *et al.* Supplementary mmc1, pg 62) [6]. It was not immediately identified, as it is the only gene in the *M.genitalium* whole-cell model genome with the protein *refolding* GO term. *GroES* (MG_393) acts as a co-chaperone for MG_392, forming the *GroEL-GroES* chaperonin complex. This was simulated using comparative gene knockouts to see if it explained the additional redundant essential relationships of MG_305 (Table 18). However, deleting pairs or triplets of genes resulted in a non-dividing *in-silico* cell and Minesweeper_256 with MG_392 and MG_393 deleted and MG_305 restored produces a non-dividing *in-silico* cell.

It appears that these genes contribute to a threshold level that is required for the cell to divide, where removing too many of them prevents division. It could be that the removal of these proteins that are involved in late protein folding (Karr *et al.* Supplementary mmc1, pg 63) [6] prevents the cell from producing the required threshold of abundance for FtsZ protein to start division (Karr *et al.* Supplementary mmc1, pg 21) [6], or perhaps earlier in the cell cycle, a lack of each of the required five chromosome segregation proteins (Karr *et al.* Supplementary mmc1, pg 34) [6] prevents successful chromosome segregation.

	Gene Deletion	Outcome	Gene Deletion	Outcome
1	MG_305	Dividing	MG_305	Dividing
2	MG_305	Dividing	MG_305	Dividing
3	MG_305	Dividing	MG_305	Dividing
4	MG_305	Dividing	MG_305	Dividing
5	MG_305	Dividing	MG_305	Dividing
6	MG_305	Dividing	MG_305	Dividing
7	MG_305	Dividing	MG_305	Dividing
8	MG_305	Dividing	MG_305	Dividing
9	MG_305	No Division	MG_305	Dividing
10	MG_305	Dividing	MG_305	Dividing
11	MG_393	Dividing	MG_392, MG_393	No Division
12	MG_393	Dividing	MG_392, MG_393	Dividing
13	MG_393	Dividing	MG_392, MG_393	Dividing
14	MG_393	Dividing	MG_392, MG_393	Dividing
15	MG_393	No Division	MG_392, MG_393	Dividing
16	MG_393	Dividing	MG_392, MG_393	Dividing
17	MG_393	Dividing	MG_392, MG_393	Dividing
18	MG_393	Dividing	MG_392, MG_393	Dividing
19	MG_393	Dividing	MG_392, MG_393	Dividing
20	MG_393	Dividing	MG_392, MG_393	Dividing
21	MG_305 and MG_393	No Division	MG_305, MG_392, MG_393	No Division
22	MG_305 and MG_393	No Division	MG_305, MG_392, MG_393	No Division
23	MG_305 and MG_393	No Division	MG_305, MG_392, MG_393	No Division
24	MG_305 and MG_393	No Division	MG_305, MG_392, MG_393	No Division
25	MG_305 and MG_393	No Division	MG_305, MG_392, MG_393	No Division
26	MG_305 and MG_393	No Division	MG_305, MG_392, MG_393	No Division
27	MG_305 and MG_393	No Division	MG_305, MG_392, MG_393	No Division
28	MG_305 and MG_393	No Division	MG_305, MG_392, MG_393	No Division
29	MG_305 and MG_393	No Division	MG_305, MG_392, MG_393	No Division
30	MG_305 and MG_393	No Division	MG_305, MG_392, MG_393	No Division
31	Minesweeper_MG305	No Division	Minesweeper_MG305	No Division
32	Minesweeper_MG305	No Division	Minesweeper_MG305	No Division
33	Minesweeper_MG305	No Division	Minesweeper_MG305	No Division
34	Minesweeper_MG305	No Division	Minesweeper_MG305	No Division
35	Minesweeper_MG305	No Division	Minesweeper_MG305	No Division
36	Minesweeper_MG305	No Division	Minesweeper_MG305	No Division
37	Minesweeper_MG305	No Division	Minesweeper_MG305	No Division
38	Minesweeper_MG305	No Division	Minesweeper_MG305	No Division
39	Minesweeper_MG305	No Division	Minesweeper_MG305	No Division
40	Minesweeper_MG305	No Division	Minesweeper_MG305	No Division
41	GAMA_MG305	No Division	GAMA_MG305	No Division
42	GAMA_MG305	No Division	GAMA_MG305	No Division
43	GAMA_MG305	No Division	GAMA_MG305	No Division
44	GAMA_MG305	No Division	GAMA_MG305	No Division

	Gene Deletion	Outcome	Gene Deletion	Outcome
45	GAMA_MG305	No Division	GAMA_MG305	No Division
46	GAMA_MG305	No Division	GAMA_MG305	No Division
47	GAMA_MG305	No Division	GAMA_MG305	No Division
48	GAMA_MG305	No Division	GAMA_MG305	No Division
49	GAMA_MG305	No Division	GAMA_MG305	No Division
50	GAMA_MG305	No Division	GAMA_MG305	No Division

Table 18. Testing potentially redundant genes using comparative gene knockouts. MG_305, MG_393, MG_305 and MG_393, MG_305 and MG_392 and MG_392 are triple, double, and single gene knockouts in an otherwise unmodified *in-silico* cell. Minesweeper_MG305 and GAMA_MG305 are Minesweeper_256 and GAMA_237 with MG_393 reintroduced (MG_392 and MG_393 in the second set) and MG_305 deleted.

4.4.8 High Essential Genes

Our comparison of the genomes also found 17 genes knocked out in GAMA_237 that have high essentiality [10] (Table 19). They were defined as essential by single knockout in an unmodified *M.genitalium* whole-cell model (Table 6) but could be removed in the genomic context of GAMA_237 without preventing division (Table 15). I also found that four of these 17 genes (MG_084, MG_295, MG_379, MG_445) could be removed as a group in the genomic context of Minesweeper_256, but doing so greatly increased the number of non-dividing cells produced (74%, simulation data is available [161]).

These 17 genes can be grouped into either transcription-related or translation-related functions. The transcription-related genes produce enzymes required for transcription in the model (Karr *et al.* Supplementary mmc1, pg 93 [6]), all of which have been removed in the GAMA_237 minimal genome. In addition, I found that the five modelled transcriptional regulators (Karr *et al.* Supplementary Table S3P col D [6]) were removed from GAMA_237. This removes the process of transcription from the *in-silico* cell.
The translation-related genes are involved in the two parts of the core translation machinery (consisting of ribosome synthesis, tRNA maturation, and tRNA aminoacylation [170]), with only tRNA aminoacylation being conserved in GAMA_237 (Karr *et al.* Supplementary mmc1, pg 104 [6]). Nine genes involved in tRNA maturation are removed (Karr *et al.* Supplementary Table S3AB [6]), and a key gene in ribosome synthesis (MG_367) is deleted (Karr *et al.* Supplementary mmc1, pg 89 [6]) from GAMA_237. This effectively removes the process of translation from the *in-silico* cell.

Gene	Annotation	Function
Transcription-related		
MG_022	*rpoE*	DNA-directed RNA polymerase subunit delta. Presence causes increased specificity of transcription, a decreased affinity for nucleic acids, and enhanced recycling.
MG_141	*nusA*	Transcription termination/antitermination protein. Participates in both.
MG_177	*rpoA*	DNA-directed RNA polymerase subunit alpha. Catalyzes the transcription of DNA into RNA using the four ribonucleoside triphosphates as substrates.
MG_249	*sigA*	RNA polymerase sigma factor. The primary initiation factor during exponential growth, promoting the attachment of RNA polymerase to specific sites.
MG_282	*greA*	Transcription elongation factor. Cleaves the fraction of nascent transcripts that get trapped at arresting sites, resuming elongation and allowing efficient RNA polymerase transcription.
MG_340	*rpoC*	DNA-directed RNA polymerase subunit beta. Catalyzes the transcription of DNA into RNA using the four ribonucleoside triphosphates as substrates.
MG_341	*rpoB*	Additional part of DNA-directed RNA polymerase subunit beta.
Translation-related		
MG_008	*mnmE*	tRNA modification GTPase. Addition of a carboxymethylaminomethyl group to certain tRNAs.
MG_084	*tilS*	tRNA(Ile)-lysidine synthase. Ligates lysine to the AUA codon-specific tRNA, changing the amino acid specificity from methionine to isoleucine.
MG_182	*truA*	tRNA pseudouridine synthase A. Forms pseudouridine in the anticodon stem and loop of tRNAs.
MG_295	*mnmA*	tRNA-specific 2-thiouridylase. Catalyzes 2-thiolation of uridine in tRNAs.
MG_347	*trmB*	tRNA methyltransferase. Catalyzes the formation of N7-methylguanine in tRNA.
MG_367	*rnc*	Ribonuclease 3. Produces ribosome large and small RNAs (23S and 16S). Processes some mRNAs and tRNAs. Digests double-stranded RNA. Other rRNA processing genes: MG_110, MG_139, MG_425 also removed in GAMA_237.
MG_372	*thiI*	tRNA sulfurtransferase. Catalyzes the transfer of sulfur to tRNAs to produce 4-thiouridine, and catalyzes the transfer of sulfur to carrier protein ThiS (a step in the synthesis of thiazole).
MG_379	*mnmG*	Forms a tetramer with MG_008. Addition of a carboxymethylaminomethyl group to certain tRNAs.
MG_445	*trmD*	tRNA methyltransferase. Specifically methylates guanosine-37 in various tRNAs.
MG_465	*rnpA*	Ribonuclease P protein component. Produces mature tRNAs (catalyzes removal of 5'-leader sequence). Additionally broadens the substrate specificity of the ribozyme through binding.

Table 19. High essential genes from GAMA_237 genomic context. Protein annotation and function obtained from UniProt [160], based on Fraser *et al.*'s *M.genitalium* G37 genome [58].

It is likely that that the lower *in-silico* division rate for GAMA_237 (33%) is due to the cell being reliant on favourable random initial conditions (i.e. already present RNAs and proteins), within the biologically feasible conditions allowed, to survive a single generation. This underlines a problem with using *in-silico* models in genome design. To the best of my knowledge, the *M.genitalium* whole-cell model is modelled correctly, including the implementation of deleting genes (Karr *et al.* Supplementary mmc1, pg 117 [6]; Simulation.m, lines 194-350 [163]). However, with the complete model only capable of modelling a single generation, *in-silico* cells that would not produce a dividing second generation *in-silico* cell (or a functioning *in-vivo* cell) can divide successfully and appear functional.

Modelling multiple generations is required to allow the production of *in-silico* genome designs that could reliably predict *in-vivo* genome function. In theory, if the final conditions of a simulation (post division) were extractable and the initial conditions could be set for simulations, the final conditions of the previous simulation could be used as the initial conditions of the next. By keeping the genetic edits constant, *M.genitalium* whole-cell model simulations could be chained together in an approximation of multiple generations. The *E.coli* whole-cell model [152] (code available, currently *in review*) models multiple generations, as should all future whole-cell models.

4.4.9 Comparison of Shared Deletions to *JCVI-Syn3.0*

I attempted to gain further insight by using BLAST to compare the shared deletions to *JCVI-Syn3.0* (Table 20) (tblastn, query: *JCVI-Syn3.0* amino acid sequence, database: nucleotide collection (nr/nt), organism: *Mycoplasma genitalium* G37 (taxid:243273)). I matched 56% of *JCVI-Syn3.0* genes to the *M.genitalium* whole-cell modelled genes, finding that 73 of the 141 shared deletions had no BLAST match with *JCVI-Syn3.0*. There were 15 deletions in common and 53 deletions not removed in *JCVI-Syn3.0*. Any conclusions drawn from this explicit comparison would be weak, as even the 56% matched *JCVI-Syn3.0* genes varied in the BLAST confidence. This is mainly due to differences between species, *JCVI-Syn3.0* is a reduction of *JCVI-Syn1.0* which is based on *Mycoplasma mycoides* not *M.genitalium*.

Deleted *in-silico* and in *JCVISyn3.0*	Deleted *in-silico*, not deleted in *JCVISyn3.0*	No tblastn match for gene deletion
15 genes	53 genes	73 genes
MG_009	MG_030	MG_012
MG_062	MG_046	MG_014
MG_063	MG_048	MG_015
MG_097	MG_050	MG_020
MG_183	MG_055	MG_027
MG_293	MG_059	MG_029
MG_339	MG_061	MG_040
MG_346	MG_072	MG_052
MG_352	MG_073	MG_064
MG_359	MG_083	MG_065
MG_391	MG_085	MG_075
MG_399	MG_109	MG_086
MG_401	MG_110	MG_101
MG_447	MG_119	MG_105
MG_498	MG_121	MG_120
	MG_122	MG_123
	MG_132	MG_127
	MG_139	MG_130
	MG_170	MG_143
	MG_172	MG_149
	MG_205	MG_184
	MG_206	MG_186
	MG_208	MG_187
	MG_209	MG_188
	MG_214	MG_189
	MG_235	MG_190
	MG_239	MG_191
	MG_240	MG_192
	MG_244	MG_200
	MG_252	MG_210
	MG_262	MG_213
	MG_265	MG_217
	MG_297	MG_218
	MG_298	MG_225
	MG_324	MG_226
	MG_329	MG_227
	MG_333	MG_236
	MG_335	MG_259
	MG_336	MG_264
	MG_369	MG_277
	MG_370	MG_288
	MG_380	MG_309
	MG_400	MG_310

Deleted *in-silico*, not deleted in *JCVISyn3.0*	No tblastn match for gene deletion
MG_402	MG_312
MG_404	MG_316
MG_405	MG_317
MG_409	MG_318
MG_421	MG_327
MG_442	MG_344
MG_457	MG_349
MG_460	MG_353
MG_463	MG_355
MG_464	MG_356
	MG_358
	MG_376
	MG_385
	MG_386
	MG_390
	MG_392
	MG_393
	MG_398
	MG_403
	MG_408
	MG_425
	MG_428
	MG_438
	MG_448
	MG_454
	MG_467
	MG_468
	MG_476
	MG_482
	MG_526

Table 20. Comparing shared gene deletions to *JCVI-Syn3.0* using tblastn. A published version of the table is available [56] .

4.5 Discussion

The group created two genome design algorithms (Minesweeper and GAMA) that used computational design-simulate-test cycles to produce *in-silico M.genitalium* minimal genomes (Minesweeper_256 and GAMA_237, 36% and 41% *in-silico* reductions respectively), producing evidence for multiple minima for *M.genitalium in-silico*. If biologically correct, the subsequent *in-vivo* minimal genome predictions are smaller than *JCVI-syn3.0* (currently the smallest genome that can be grown in pure culture at 473 genes [12]) and smaller than the most recent predictions for a reduced *Mycoplasma*

genome (413 genes) [13]. However, these predicted genomes have not been tested for successful growth and division over multiple generations. In addition to predicting these greatly reduced predicted genomes, I identified 10 low essential genes [10] (Table 18).

Specific issues have been highlighted, including the modelling of *DisA*, and the biologically-infeasible removal of the high essential genes *in-silico* (an outcome of the model's single generation lifespan). I do not have confidence in the *in-silico* high essential or *DisA* gene deletions.

There are limitations to the approach presented here. Models are not perfect representations of reality. Through necessity, the *M.genitalium* whole-cell model bases some of its parameters on data from other bacteria [6]. Additionally, complete multi-generation simulations are not possible with the *M.genitalium* whole-cell model (the only whole-cell model available at the time), and *M.genitalium* has genes of unknown function that the model cannot account for. The success of our *in-silico* genomes *in-vivo* will be dependent on the accuracy of the model, which is untested at this large-scale number of genetic modifications.

However, the gene deletions shared by GAMA_237 and Minesweeper_256 (141 gene deletions, Table 15) and the deletions responsible for each of the phosphate "pivot points" are worthy of *in-vivo* testing. It remains possible that these could predict a viable *M.genitalium* minimal cell. Given that the impact of the unmodelled genes is unknown (e.g. if they perform a unique essential function with a gene / gene product that has already been removed, then the *in-vivo* cell will not survive) until these predictions are tested experimentally we cannot firmly state how long the predicted reduced *in-vivo* cells would survive and replicate, and whether they represent a truly minimal *M.genitalium* genome.

Chapter 5 - Testing Theoretical Minimal Genomes *in-silico*

5.1 Statement of Collaboration

Sections of this chapter have previously been published with the author of this thesis as the lead author. The sections reproduced here are solely this author's work, with guidance from the supervisors.

- Rees-Garbutt, J., Grierson, C. & Marucci, L. Testing theoretical minimal genomes using whole-cell models. *bioRxiv.* doi:10.1101/2020.03.26.010363 (2020).
    - Lead author

5.2 Aims

Numerous authors have pondered what the minimal gene set for life might be (Section 1.3.2.c) and many hypothetical minimal gene sets have been proposed, including at least 10 for *M.genitalium* [6,55,57,59–61,74–77]. None of these have been reported to be tested *in-vivo* or *in-silico*. *In-vivo* testing would be extremely difficult as *M.genitalium* is very difficult to grow in the laboratory and laborious to engineer [11,12]. However, having the *M.genitalium* model running provided an excellent opportunity to test them *in-silico*. I edited and ran versions of the *M.genitalium* whole-cell model equivalent to the eight smallest hypothetical minimal gene sets from the literature. None of these initially produced *in-silico* cells that could grow and divide. Taking into account knowledge that I had accumulated about the *M.genitalium* whole-cell model genome and the functions of its genes, I was able to repair these sets by reintroducing selected genes, so that they produced dividing *in-silico* cells.

5.3 Results

5.3.1 Adapting the Minimal Gene Sets to the *M.genitalium* whole-cell Model

Ten minimal gene sets were found in the literature that were designed with *M.genitalium* genes [6,55,57,59–61,74–77], however two sets [57,77] were excluded as they were considered derivative of the Gil 2014 set [55] being identical apart from four genes in the Shuler *et al.* set (MG_056, MG_146, MG_388, MG_391) and four genes not in the Gil *et al.* 2004 set (MG_009, MG_091, MG_132, MG_460).

Minimal Gene Set	Code Name	Design Methodology
Mushegian and Koonin 1996 [59]	Bethesda	Comparative Genomics
Hutchison *et al.* 1999 [60]	Rockville	Single Gene Deletions
Tomita *et al.* 1999 [74]	Fujisawa	Protocell
Glass *et al.* 2006 [61]	Rockville 2	Single Gene Deletions
Forster and Church 2006 [75]	Nashville	Protocell
Karr *et al.* 2012 [6]	Stanford	Single Gene Deletions
Huang *et al.* 2013 [76]	Guelph	Comparative Genomics
Gil 2014 [55]	Valencia	Comparative Genomics

Table 21. Code names for the minimal gene sets from the literature.

To prevent confusion, I named the sets after the main location where the set was constructed (Table 21). The Bethesda set is a comparison of *M.genitalium* protein sequences to *Haemophilus influenzae* (representatives of gram-positive and gram-negative bacteria respectively) [59]. The Rockville set is the result of applying global transposon mutagenesis to *M.genitalium in-vivo* to identify non-essential genes [60]. The Fujisawa set is an *in-silico* model of a hypothetical cell constructed from 127 *M.genitalium* genes using the E-Cell software [74]. The Rockville 2 set is an expansion of the original global transposon mutagenesis research on *M.genitalium*, with properly conducted isolation and characterisation of pure clonal populations [61]. The Nashville set is a list of 151 *E.coli* genes (compared to *M.genitalium* genes within the paper) to produce a chemical system capable of replication and evolution [75] . The Stanford set is the result of *in-silico* single gene knockouts conducted using the *M.genitalium* whole-cell model [6]. The Guelph set is the result of a comparative genomics analysis of 186 bacterial genomes [76]. The Valencia set compared the genome of *M.genitalium* with genetic data of five insect endosymbionts [55] .

To begin testing the eight minimal gene sets to see if they produced *in-silico* dividing cells, I had first to adapt each set for use in simulations, removing any genes unmodelled in the *M.genitalium* whole-cell model. The sets Nashville, Fujisawa and Stanford are as they appear in the literature, the others needed adapting by removing genes as follows (Table 22). Guelph had seven genes removed: six because they are not in the whole-cell model and one copy of MG_231 was removed as it was originally listed twice. Valencia had eight genes removed: seven that are not present in the

M.genitalium whole-cell model and one copy of MG_231 was removed as it was originally listed twice. Bethesda had 15 genes removed: 13 as they are not in the whole-cell model and MG_297 and MG_336 were reduced to a single copy each as they were both originally listed twice. Rockville had 41 genes removed as they are not in the whole-cell model. Rockville 2 had 44 genes removed as they were not in the whole-cell model.

As expected, the minimal gene sets designed as protocells (Nashville, Fujisawa) have the smallest predicted *in-silico* genome size. Of the comparative genomics minimal gene sets, Guelph is substantially smaller than Valencia and Bethesda due to comparing 186 bacterial species for common genes [76], with Valencia only six species [55], and Bethesda directly comparing two species [59]. The single gene deletion minimal gene sets (Stanford, Rockville, Rockville 2) have similar numbers of *in-vivo* deletions, but Rockville and Rockville 2 have the highest numbers of genes that are missing from the *M.genitalium* whole-cell model. This is due to the nature of exploratory genetic work, genes can be disrupted *in-vivo* whether or not the gene function is known. To be implemented *in-silico*, however, the function of the genes also must be known. All the genes in Stanford are contained in the whole-cell model, as the single gene deletions were conducted *in-silico* using the model and so did not target unmodelled genes. The gene content of the minimal gene sets (Table 23) and the required gene deletions to produce these minimal gene sets in the *M.genitalium* whole-cell model (Table 24) are listed.

	Design approach	*in-vivo* genome design size*	Unmodelled genes^ in genome design	Single *in-vivo* gene deletions*	Unmodelled genes^ in single gene deletions	Predicted *in-silico* genome size*	Predicted gene deletions *in-silico**
Nashville	Protocell	89	0	-	-	89	270
Fujisawa	Protocell	98	0	-	-	98	261
Guelph	Comparative Genomics	123	5	-	-	118	241
Valencia	Comparative Genomics	180	6	-	-	174	185
Bethesda	Comparative Genomics	253	12	-	-	241	118
Stanford	Single Gene Deletions	-	-	117	0	242	117
Rockville 2	Single Gene Deletions	-	-	101	44	302	57
Rockville	Single Gene Deletions	-	-	94	41	306	53
M.genitalium whole-cell Model*	-	-	124	-	-	359	-
M.genitalium in-vivo *	-	483		-	-	-	-

Table 22. Minimal gene sets from the literature, compared with *M.genitalium in-vivo* and the whole-cell model. *M.genitalium* has 42 RNA-coding genes that are not included in this table. * = protein-coding genes. ^ = due to unknown function.

	Bethesda	Rockville	Fujisawa	Rockville 2	Nashville	Stanford	Guelph	Valencia	Common
MG_001	I	I		I		I	I	I	I
MG_002						I			
MG_003	I	I		I		I	I	I	I
MG_004	I	I		I		I	I	I	I
MG_005	I	I	I	I	I	I	I	I	I
MG_006	I	I		I		I		I	
MG_007		I		I		I		I	
MG_008	I	I		I		I	I	I	I
MG_009	I								
MG_010						I			
MG_011						I			
MG_012	I	I				I			
MG_013	I	I		I		I			
MG_014				I					
MG_015	I	I		I					
MG_018						I			
MG_019	I	I		I		I	I	I	I
MG_020		I		I		I			
MG_021	I	I	I	I	I	I	I	I	I
MG_022		I		I		I			
MG_023	I	I	I	I		I		I	
MG_024	I	I				I	I	I	
MG_025				I		I			
MG_026	I	I		I	I	I		I	
MG_027		I		I					
MG_028		I		I		I			
MG_029				I					
MG_030	I	I		I				I	
MG_031		I		I		I		I	
MG_032						I			
MG_033	I		I						
MG_034		I		I		I			
MG_035	I	I	I	I	I	I	I	I	I
MG_036	I	I	I	I	I	I	I	I	I
MG_037		I		I		I			
MG_038	I	I	I	I		I			
MG_039		I							
MG_040		I							
MG_041		I	I	I		I		I	
MG_042	I	I		I		I			
MG_043	I	I		I		I			
MG_044	I	I		I		I			
MG_045	I	I		I		I			
MG_046		I		I		I	I	I	

	Bethesda	Rockville	Fujisawa	Rockville 2	Nashville	Stanford	Guelph	Valencia	Common
MG_047	I	I		I		I	I	I	I
MG_048	I	I		I		I	I	I	I
MG_049	I			I		I			
MG_050	I	I		I					
MG_051						I			
MG_052	I			I					
MG_053	I	I		I		I			
MG_054	I	I		I		I	I	I	I
MG_055		I		I				I	
MG_473		I		I		I			
MG_474		I		I		I			
MG_056	I	I				I	I	I	
MG_057		I		I		I			
MG_058	I	I		I		I	I	I	I
MG_059	I	I		I			I	I	
MG_060	I	I		I		I			
MG_061		I							
MG_062									
MG_063	I	I							
MG_064		I		I					
MG_065	I	I		I					
MG_066	I	I				I	I	I	
MG_067						I			
MG_068		I		I		I			
MG_069		I	I	I		I		I	
MG_070	I	I	I	I	I	I	I	I	I
MG_071	I	I		I		I			
MG_072	I	I		I		I	I	I	I
MG_073	I	I		I					
MG_074		I		I		I			
MG_075		I		I					
MG_076		I		I		I			
MG_077	I	I		I		I			
MG_078	I	I		I		I			
MG_079	I	I		I		I			
MG_080	I	I		I		I			
MG_081	I	I	I	I	I	I	I	I	I
MG_082	I	I	I	I	I	I	I	I	I
MG_083	I	I		I			I	I	
MG_084		I		I		I	I	I	
MG_085				I					
MG_086	I	I		I					
MG_087	I	I	I	I	I	I	I	I	I
MG_088	I	I	I	I	I	I	I	I	I
MG_089	I	I	I	I	I	I	I	I	I

	Bethesda	Rockville	Fujisawa	Rockville 2	Nashville	Stanford	Guelph	Valencia	Common
MG_090	I	I	I	I	I	I		I	
MG_091	I	I		I		I			
MG_092	I	I	I	I	I	I	I	I	I
MG_093	I	I	I	I	I	I		I	
MG_094	I	I		I		I	I	I	I
MG_095		I		I		I			
MG_096						I			
MG_097	I	I		I				I	
MG_098		I		I		I			
MG_099		I		I	I	I			
MG_100		I		I	I	I			
MG_101		I		I					
MG_102	I	I		I		I	I	I	I
MG_103						I			
MG_476		I		I		I			
MG_104	I	I		I		I			
MG_105		I		I					
MG_106	I	I		I		I			
MG_107	I	I		I		I	I	I	I
MG_108		I		I		I			
MG_109		I		I		I			
MG_110						I			
MG_111	I	I	I	I		I	I	I	I
MG_112	I	I				I	I	I	
MG_113	I	I	I	I	I	I	I	I	I
MG_114	I	I	I			I			
MG_115		I				I			
MG_116		I				I			
MG_117		I		I		I			
MG_118	I	I		I		I			
MG_119	I	I		I					
MG_120	I	I		I					
MG_121		I							
MG_122	I	I		I		I			
MG_123		I		I					
MG_124	I	I		I		I	I	I	I
MG_125	I	I		I		I			
MG_126	I	I	I	I	I	I	I	I	I
MG_127	I	I		I					
MG_128		I		I		I			
MG_129		I		I		I			
MG_130				I					
MG_131						I			
MG_132				I					
MG_133		I		I		I			

	Bethesda	Rockville	Fujisawa	Rockville 2	Nashville	Stanford	Guelph	Valencia	Common
MG_134		I				I			
MG_135		I		I		I			
MG_136	I	I	I	I	I	I	I	I	I
MG_137		I		I		I			
MG_138	I	I				I	I	I	
MG_139		I		I		I			
MG_140						I			
MG_141	I	I		I		I	I	I	I
MG_477				I		I			
MG_142	I	I	I	I	I	I	I	I	I
MG_143	I	I		I		I		I	
MG_144		I		I		I			
MG_145	I	I		I		I	I	I	I
MG_146		I		I		I			
MG_147		I		I		I			
MG_148		I		I		I			
MG_149									
MG_478		I				I			
MG_150	I	I	I	I	I	I	I	I	I
MG_151	I	I	I	I	I	I	I	I	I
MG_152	I	I	I	I	I	I	I	I	I
MG_153	I	I	I	I	I	I		I	
MG_154	I	I	I	I	I	I	I	I	I
MG_155	I	I	I	I	I	I	I	I	I
MG_156	I	I	I	I	I	I	I	I	I
MG_157	I	I	I	I	I	I	I	I	I
MG_158	I	I	I	I	I	I	I	I	I
MG_159	I	I	I	I	I	I		I	
MG_160	I	I	I	I	I	I	I	I	I
MG_161	I	I	I	I	I	I	I	I	I
MG_162	I	I	I	I	I	I	I	I	I
MG_163	I	I	I	I	I	I	I	I	I
MG_164	I	I	I	I	I	I		I	
MG_165	I	I	I	I	I	I	I	I	I
MG_166	I	I	I	I	I	I	I	I	I
MG_167	I	I	I	I	I	I	I	I	I
MG_168	I	I	I	I	I	I	I	I	I
MG_169	I	I		I	I	I		I	
MG_170	I	I		I		I	I	I	I
MG_171	I	I		I	I	I	I	I	I
MG_172	I	I		I		I	I	I	I
MG_173	I	I	I	I	I	I	I	I	I
MG_174	I	I	I	I	I	I		I	
MG_175	I	I	I	I	I	I	I	I	I
MG_176	I	I	I	I	I	I	I	I	I

	Bethesda	Rockville	Fujisawa	Rockville 2	Nashville	Stanford	Guelph	Valencia	Common
MG_177	I	I	I	I		I	I	I	I
MG_178	I	I	I	I	I	I	I	I	I
MG_179		I		I		I			
MG_180	I	I		I		I			
MG_181		I		I		I			
MG_182	I			I		I			
MG_183						I			
MG_184		I		I		I			
MG_185						I			
MG_186				I					
MG_187	I	I		I					
MG_188		I		I					
MG_189		I		I					
MG_190		I		I					
MG_191				I		I			
MG_192				I		I			
MG_194	I	I	I	I	I	I	I	I	I
MG_195	I	I		I	I	I	I	I	I
MG_196	I	I	I	I	I	I	I	I	I
MG_197	I	I	I	I	I	I		I	
MG_198	I	I	I	I	I	I	I	I	I
MG_199		I		I		I			
MG_200		I		I					
MG_201	I	I		I		I		I	
MG_202				I		I			
MG_203	I	I		I					
MG_204	I	I		I		I			
MG_205		I		I					
MG_206	I	I		I					
MG_207						I			
MG_208		I		I		I			
MG_209	I			I					
MG_210	I	I				I			
MG_480		I		I		I			
MG_481		I		I		I			
MG_211		I		I		I			
MG_482		I		I					
MG_212	I	I		I		I		I	
MG_213									
MG_214		I							
MG_215	I	I	I	I		I		I	
MG_216	I	I	I	I		I		I	
MG_217		I		I		I			
MG_218		I		I		I			
MG_491		I		I		I			

	Bethesda	Rockville	Fujisawa	Rockville 2	Nashville	Stanford	Guelph	Valencia	Common
MG_219		I		I		I			
MG_220		I				I			
MG_221	I	I		I		I			
MG_222	I	I		I		I	I	I	
MG_223		I		I		I			
MG_224	I	I		I		I	I	I	
MG_225		I		I					
MG_226									
MG_227	I	I						I	
MG_228	I	I		I		I		I	
MG_229	I	I		I		I		I	
MG_230		I		I		I			
MG_231	I	I		I		I	I	I	I
MG_232	I	I	I	I	I	I		I	
MG_233		I		I		I			
MG_234	I	I	I	I	I	I	I	I	I
MG_235		I		I				I	
MG_236		I		I					
MG_237						I			
MG_238	I	I				I			
MG_239	I	I		I		I			
MG_240		I		I					
MG_241		I		I		I			
MG_242		I		I		I			
MG_243		I		I		I			
MG_244	I	I							
MG_245	I	I		I		I			
MG_246		I		I		I			
MG_247	I	I		I		I			
MG_248		I				I			
MG_249	I	I	I	I		I	I	I	I
MG_250	I	I		I			I	I	
MG_251	I	I	I	I	I	I		I	
MG_252	I	I							
MG_253	I	I	I	I	I	I	I	I	I
MG_254	I	I		I		I	I	I	I
MG_255						I			
MG_494						I			
MG_256						I			
MG_257	I	I	I	I	I	I	I	I	I
MG_258	I	I		I	I	I	I	I	I
MG_259	I	I		I	I			I	
MG_260		I				I			
MG_261	I	I		I		I	I	I	I
MG_262	I	I		I			I	I	

	Bethesda	Rockville	Fujisawa	Rockville 2	Nashville	Stanford	Guelph	Valencia	Common
MG_498	I	I							
MG_263		I		I		I			
MG_264								I	
MG_265	I	I		I					
MG_266	I	I	I	I	I	I	I	I	I
MG_267		I		I		I			
MG_268	I					I			
MG_269						I			
MG_270	I	I		I		I			
MG_271	I	I				I			
MG_272	I	I		I		I			
MG_273	I	I		I		I			
MG_274	I	I		I		I			
MG_275	I	I		I		I			
MG_276	I	I		I		I			
MG_277		I		I		I			
MG_278	I	I		I		I			
MG_279						I			
MG_280						I			
MG_281						I			
MG_282	I	I		I		I	I	I	I
MG_283	I	I	I	I	I	I		I	
MG_284		I				I			
MG_285						I			
MG_286						I			
MG_287	I	I		I		I			
MG_288									
MG_289		I							
MG_290		I							
MG_291									
MG_505		I		I		I			
MG_292	I	I	I	I	I	I	I	I	I
MG_293	I								
MG_294						I			
MG_295	I			I		I	I	I	
MG_296				I		I			
MG_297	I	I		I		I	I	I	I
MG_298		I							
MG_299	I	I		I		I			
MG_300	I	I	I	I		I	I	I	I
MG_301	I	I	I	I		I	I	I	I
MG_302		I		I		I			
MG_303		I		I		I			
MG_304		I		I		I			
MG_305	I	I		I		I	I	I	I

	Bethesda	Rockville	Fujisawa	Rockville 2	Nashville	Stanford	Guelph	Valencia	Common
MG_306		I		I		I			
MG_307		I		I		I			
MG_308				I		I			
MG_309		I		I					
MG_310				I					
MG_311		I	I	I	I	I	I	I	
MG_312		I		I		I			
MG_313		I		I		I			
MG_314		I		I		I			
MG_315		I				I		I	
MG_316									
MG_317				I		I			
MG_318		I		I		I			
MG_319		I		I		I			
MG_320		I		I		I			
MG_321		I		I		I			
MG_322	I	I		I		I			
MG_323		I		I		I			
MG_515				I		I			
MG_324		I		I					
MG_325	I	I	I	I	I	I		I	
MG_326		I		I		I			
MG_327				I					
MG_328						I			
MG_329		I		I		I	I	I	
MG_330	I	I		I		I			
MG_331		I		I		I			
MG_332	I	I		I		I			
MG_333	I	I		I					
MG_334	I	I	I	I	I	I	I	I	I
MG_335	I			I		I			
MG_516		I		I		I			
MG_517		I		I		I			
MG_336	I	I		I				I	
MG_337		I		I		I			
MG_338				I		I			
MG_339	I								
MG_340	I	I	I	I		I	I	I	I
MG_341	I	I	I	I		I	I	I	I
MG_342		I		I		I			
MG_343						I			
MG_344		I	I	I					
MG_345	I		I	I	I	I	I	I	
MG_346	I							I	
MG_347	I	I		I		I			

	Bethesda	Rockville	Fujisawa	Rockville 2	Nashville	Stanford	Guelph	Valencia	Common
MG_348		I		I		I			
MG_349		I		I					
MG_350		I		I		I			
MG_521		I		I		I			
MG_351	I	I	I	I	I	I		I	
MG_352									
MG_353	I	I		I				I	
MG_354		I		I		I			
MG_355	I	I							
MG_356		I		I					
MG_357	I	I		I		I			
MG_358	I	I							
MG_359	I	I							
MG_360		I				I			
MG_361	I	I	I	I	I	I	I	I	I
MG_362	I	I	I	I	I	I	I	I	I
MG_363	I	I	I	I	I	I		I	
MG_522	I	I	I	I	I	I		I	
MG_364		I		I		I			
MG_365	I	I	I	I	I	I			
MG_366				I		I			
MG_367	I	I				I	I	I	
MG_368		I		I		I			
MG_369		I		I					
MG_370									
MG_371		I		I		I			
MG_372				I		I			
MG_373		I		I		I			
MG_374		I		I		I			
MG_375	I	I	I	I	I	I		I	
MG_376		I		I					
MG_377		I		I		I			
MG_378	I	I	I	I	I	I	I	I	I
MG_379	I	I		I		I		I	
MG_380	I								
MG_381		I		I		I			
MG_382	I	I		I		I			
MG_383	I	I		I		I			
MG_384	I	I		I		I	I	I	I
MG_524		I		I		I			
MG_385									
MG_386		I		I		I			
MG_387	I	I		I		I		I	
MG_388		I		I		I		I	
MG_389		I		I		I			

	Bethesda	Rockville	Fujisawa	Rockville 2	Nashville	Stanford	Guelph	Valencia	Common
MG_390									
MG_391	I	I		I		I		I	
MG_392	I	I		I	I	I	I	I	I
MG_393	I	I		I	I	I		I	
MG_394	I			I		I	I	I	
MG_395				I		I			
MG_396		I		I		I		I	
MG_397		I				I			
MG_398	I	I						I	
MG_399	I	I		I			I	I	
MG_400	I	I		I			I	I	
MG_401	I	I		I			I	I	
MG_402	I	I		I				I	
MG_403	I	I		I				I	
MG_404	I	I		I				I	
MG_405	I	I		I				I	
MG_406	I	I		I		I			
MG_407	I	I	I	I		I	I	I	I
MG_408	I								
MG_409		I		I					
MG_410	I								
MG_411	I								
MG_412									
MG_414						I			
MG_525						I			
MG_417	I	I	I	I	I	I	I	I	I
MG_418	I	I	I	I	I	I	I	I	I
MG_419		I		I		I			
MG_421	I			I					
MG_422		I		I		I			
MG_423				I		I			
MG_424	I	I	I	I	I	I	I	I	I
MG_425	I	I		I		I	I	I	I
MG_426	I		I	I	I	I		I	
MG_427		I		I					
MG_428									
MG_429		I	I	I		I	I	I	
MG_430	I	I	I	I		I		I	
MG_431	I	I	I	I		I	I	I	I
MG_432		I		I		I			
MG_433	I	I	I	I	I	I	I	I	I
MG_434	I	I		I		I			
MG_435	I	I		I	I	I	I	I	I
MG_437	I	I	I			I		I	
MG_438									

	Bethesda	Rockville	Fujisawa	Rockville 2	Nashville	Stanford	Guelph	Valencia	Common
MG_439		I		I		I			
MG_440				I		I			
MG_441		I		I		I			
MG_442				I		I			
MG_443		I		I		I			
MG_444	I	I	I	I	I	I	I	I	I
MG_445	I	I		I	I	I	I		
MG_446	I	I	I	I	I	I	I	I	I
MG_447		I		I					
MG_448	I	I		I					
MG_449	I					I			
MG_450		I		I		I			
MG_451	I	I	I	I	I	I	I	I	I
MG_452						I			
MG_453	I	I		I		I			
MG_454		I		I					
MG_455	I	I	I	I	I	I	I	I	I
MG_456		I				I			
MG_457	I	I		I			I	I	
MG_458	I	I		I		I		I	
MG_459		I		I		I			
MG_460		I	I	I					
MG_461		I		I		I			
MG_462	I	I	I	I	I	I	I	I	I
MG_463	I	I		I			I	I	
MG_464		I		I		I		I	
MG_465	I	I		I	I	I		I	
MG_466	I	I	I	I	I	I		I	
MG_467				I					
MG_468				I					
MG_526		I		I					
MG_469	I	I		I		I	I		
MG_470				I					

Table 23. Comparing the gene content of the minimal gene sets. Light grey genes are unmodelled in the *M.genitalium* whole-cell model. Dark grey genes cause the simulation to crash when deleted on our implementation (using Matlab R2013B on University of Bristol BlueGem's supercomputer (Section 3.4)). I = gene included in the minimal gene set. Common = genes that the minimal gene sets agree upon, but excluding the protocell designed sets.

	Bethesda	Rockville	Fujisawa	Rockville 2	Nashville	Stanford	Guelph	Valencia	Agreed
MG_001			x		x				
MG_002		x		x					
MG_003			x		x				
MG_004			x		x				
MG_005									
MG_006			x		x		x		
MG_007	x		x		x		x		
MG_008			x		x				
MG_009		x	x	x	x	x	x	x	
MG_010		x		x					
MG_011		x		x					
MG_012			x	x	x		x	x	
MG_013			x		x		x	x	
MG_014	x	x	x		x	x	x	x	
MG_015			x		x	x	x	x	
MG_018		x		x					
MG_019			x		x				
MG_020	x		x		x		x	x	
MG_021									
MG_022	x		x		x		x	x	
MG_023					x		x		
MG_024				x					
MG_025		x							
MG_026			x				x		
MG_027	x		x		x	x	x	x	
MG_028									
MG_029	x	x	x		x	x	x	x	
MG_030			x		x	x	x		
MG_031	x		x		x		x		
MG_032		x		x					
MG_033		x		x	x	x	x	x	
MG_034	x		x		x		x	x	
MG_035									
MG_036									
MG_037	x		x		x		x	x	
MG_038					x		x	x	
MG_039	x		x	x	x	x	x	x	
MG_040	x		x	x	x	x	x	x	
MG_041	x				x		x		
MG_042			x		x		x	x	
MG_043			x		x		x	x	
MG_044			x		x		x	x	
MG_045			x		x		x	x	
MG_046	x		x		x				
MG_047			x		x				

	Bethesda	Rockville	Fujisawa	Rockville 2	Nashville	Stanford	Guelph	Valencia	Agreed
MG_048			x		x				
MG_049		x	x		x		x	x	
MG_050			x		x	x	x	x	
MG_051	x	x	x	x	x		x	x	
MG_052		x	x		x	x	x	x	
MG_053			x		x		x	x	
MG_054									
MG_055	x		x		x	x	x		
MG_473	x		x		x		x	x	
MG_474									
MG_056				x					
MG_057									
MG_058			x		x				
MG_059			x		x	x			
MG_060									
MG_061	x		x	x	x	x	x	x	
MG_062	x	x	x	x	x	x	x	x	x
MG_063			x	x	x	x	x	x	
MG_064	x		x		x	x	x	x	
MG_065			x		x	x	x	x	
MG_066			x	x	x				
MG_067		x		x					
MG_068									
MG_069	x				x		x		
MG_070									
MG_071			x		x		x	x	
MG_072			x		x				
MG_073			x		x	x	x	x	
MG_074									
MG_075	x		x		x	x	x	x	
MG_076									
MG_077			x		x		x	x	
MG_078			x		x		x	x	
MG_079			x		x		x	x	
MG_080			x		x		x	x	
MG_081									
MG_082									
MG_083			x		x	x			
MG_084	x		x		x				
MG_085	x	x	x		x	x	x	x	
MG_086			x		x	x	x	x	
MG_087									
MG_088									
MG_089									
MG_090							x		

	Bethesda	Rockville	Fujisawa	Rockville 2	Nashville	Stanford	Guelph	Valencia	Agreed
MG_091			x		x		x	x	
MG_092									
MG_093							x		
MG_094			x		x				
MG_095									
MG_096		x		x					
MG_097			x		x	x	x		
MG_098	x		x		x		x	x	
MG_099	x		x				x	x	
MG_100	x		x				x	x	
MG_101	x		x		x	x	x	x	
MG_102			x		x				
MG_103		x		x					
MG_476	x		x		x		x	x	
MG_104			x		x		x	x	
MG_105	x		x		x	x	x	x	
MG_106			x		x		x	x	
MG_107			x		x				
MG_108									
MG_109	x		x		x		x	x	
MG_110	x	x	x	x	x		x	x	
MG_111					x				
MG_112			x	x	x				
MG_113									
MG_114				x	x		x	x	
MG_115				x					
MG_116				x					
MG_117									
MG_118			x		x		x	x	
MG_119			x		x	x	x	x	
MG_120			x		x	x	x	x	
MG_121	x		x	x	x	x	x	x	
MG_122			x		x		x	x	
MG_123	x		x		x	x	x	x	
MG_124			x		x				
MG_125									
MG_126									
MG_127			x		x	x	x	x	
MG_128	x		x		x		x	x	
MG_129									
MG_130	x	x	x		x	x	x	x	
MG_131		x		x					
MG_132	x	x	x		x	x	x	x	
MG_133									
MG_134				x					

	Bethesda	Rockville	Fujisawa	Rockville 2	Nashville	Stanford	Guelph	Valencia	Agreed
MG_135									
MG_136									
MG_137	x		x		x		x	x	
MG_138				x					
MG_139	x		x		x		x	x	
MG_140		x		x					
MG_141			x		x				
MG_477		x							
MG_142									
MG_143			x		x		x		
MG_144									
MG_145			x		x				
MG_146									
MG_147									
MG_148									
MG_149	x	x	x	x	x	x	x	x	x
MG_478				x					
MG_150									
MG_151									
MG_152									
MG_153							x		
MG_154									
MG_155									
MG_156									
MG_157									
MG_158									
MG_159							x		
MG_160									
MG_161									
MG_162									
MG_163									
MG_164							x		
MG_165									
MG_166									
MG_167									
MG_168									
MG_169			x				x		
MG_170			x		x				
MG_171			x						
MG_172			x		x				
MG_173									
MG_174							x		
MG_175									
MG_176									
MG_177					x				

	Bethesda	Rockville	Fujisawa	Rockville 2	Nashville	Stanford	Guelph	Valencia	Agreed
MG_178									
MG_179	x		x		x		x	x	
MG_180			x		x		x	x	
MG_181	x		x		x		x	x	
MG_182		x	x		x		x	x	
MG_183	x	x	x	x	x		x	x	
MG_184	x		x		x		x	x	
MG_185		x		x					
MG_186	x	x	x		x	x	x	x	
MG_187			x		x	x	x	x	
MG_188	x		x		x	x	x	x	
MG_189	x		x		x	x	x	x	
MG_190	x		x		x	x	x	x	
MG_191	x	x	x		x		x	x	
MG_192	x	x	x		x		x	x	
MG_194									
MG_195			x						
MG_196									
MG_197							x		
MG_198									
MG_199									
MG_200	x		x		x	x	x	x	
MG_201			x		x		x		
MG_202		x							
MG_203			x		x	x	x	x	
MG_204			x		x		x	x	
MG_205	x		x		x	x	x	x	
MG_206			x		x	x	x	x	
MG_207		x		x					
MG_208	x		x		x		x	x	
MG_209		x	x		x	x	x	x	
MG_210			x	x	x		x	x	
MG_480									
MG_481	x		x		x		x	x	
MG_211									
MG_482	x		x		x	x	x	x	
MG_212			x		x		x		
MG_213	x	x	x	x	x	x	x	x	x
MG_214	x		x	x	x	x	x	x	
MG_215					x		x		
MG_216					x		x		
MG_217	x		x		x		x	x	
MG_218	x		x		x		x	x	
MG_491									
MG_219									

	Bethesda	Rockville	Fujisawa	Rockville 2	Nashville	Stanford	Guelph	Valencia	Agreed
MG_220				x					
MG_221			x		x		x	x	
MG_222									
MG_223									
MG_224			x		x				
MG_225	x		x		x	x	x	x	
MG_226	x	x	x	x	x	x	x	x	x
MG_227			x	x	x	x	x		
MG_228			x		x		x		
MG_229			x		x		x		
MG_230	x		x		x		x	x	
MG_231			x		x				
MG_232							x		
MG_233									
MG_234									
MG_235	x		x		x	x	x		
MG_236	x		x		x	x	x	x	
MG_237		x		x					
MG_238			x	x	x		x	x	
MG_239			x		x		x	x	
MG_240	x		x		x	x	x	x	
MG_241									
MG_242									
MG_243									
MG_244			x	x	x	x	x	x	
MG_245			x		x		x	x	
MG_246									
MG_247									
MG_248				x					
MG_249					x				
MG_250			x		x	x			
MG_251							x		
MG_252			x	x	x	x	x	x	
MG_253									
MG_254			x		x				
MG_255		x		x					
MG_494		x		x					
MG_256		x		x					
MG_257									
MG_258			x						
MG_259			x			x	x		
MG_260				x					
MG_261			x		x				
MG_262			x		x	x			
MG_498			x	x	x	x	x	x	

	Bethesda	Rockville	Fujisawa	Rockville 2	Nashville	Stanford	Guelph	Valencia	Agreed
MG_263									
MG_264	x	x	x	x	x	x	x		
MG_265			x		x	x	x	x	
MG_266									
MG_267									
MG_268		x		x					
MG_269		x		x					
MG_270			x		x		x	x	
MG_271			x	x	x		x	x	
MG_272			x		x		x	x	
MG_273			x		x		x	x	
MG_274			x		x		x	x	
MG_275			x		x		x	x	
MG_276			x		x		x	x	
MG_277	x		x		x		x	x	
MG_278			x		x		x	x	
MG_279		x		x					
MG_280		x		x					
MG_281		x		x					
MG_282			x		x				
MG_283							x		
MG_284				x					
MG_285		x		x					
MG_286		x		x					
MG_287			x		x		x	x	
MG_288	x	x	x	x	x	x	x	x	x
MG_289	x		x	x	x	x	x	x	
MG_290	x		x	x	x	x	x	x	
MG_291	x	x	x	x	x	x	x	x	x
MG_505									
MG_292									
MG_293		x	x	x	x	x	x	x	
MG_294		x		x					
MG_295		x	x		x				
MG_296		x							
MG_297			x		x				
MG_298	x		x	x	x	x	x	x	
MG_299			x		x		x	x	
MG_300					x				
MG_301					x				
MG_302	x		x		x		x	x	
MG_303	x		x		x		x	x	
MG_304	x		x		x		x	x	
MG_305			x		x				
MG_306									

	Bethesda	Rockville	Fujisawa	Rockville 2	Nashville	Stanford	Guelph	Valencia	Agreed
MG_307									
MG_308		x							
MG_309	x		x		x	x	x	x	
MG_310	x	x	x		x	x	x	x	
MG_311	x								
MG_312	x		x		x		x	x	
MG_313									
MG_314									
MG_315	x		x	x	x		x		
MG_316	x	x	x	x	x	x	x	x	x
MG_317	x	x	x		x		x	x	
MG_318	x		x		x		x	x	
MG_319									
MG_320									
MG_321	x		x		x		x	x	
MG_322			x		x		x	x	
MG_323	x		x		x		x	x	
MG_515		x							
MG_324	x		x		x	x	x	x	
MG_325							x		
MG_326									
MG_327	x	x	x		x	x	x	x	
MG_328		x		x					
MG_329	x		x		x				
MG_330			x		x		x	x	
MG_331									
MG_332									
MG_333			x		x	x	x	x	
MG_334									
MG_335		x	x		x		x	x	
MG_516									
MG_517	x		x		x		x	x	
MG_336			x		x	x	x		
MG_337									
MG_338		x							
MG_339		x	x	x	x	x	x	x	
MG_340					x				
MG_341					x				
MG_342	x		x		x		x	x	
MG_343		x		x					
MG_344	x				x	x	x	x	
MG_345		x							
MG_346		x	x	x	x	x	x		
MG_347			x		x		x	x	
MG_348									

	Bethesda	Rockville	Fujisawa	Rockville 2	Nashville	Stanford	Guelph	Valencia	Agreed
MG_349	x		x		x	x	x	x	
MG_350									
MG_521									
MG_351							x		
MG_352	x	x	x	x	x	x	x	x	x
MG_353			x		x	x	x		
MG_354									
MG_355			x	x	x	x	x	x	
MG_356	x		x		x	x	x	x	
MG_357			x		x		x	x	
MG_358			x	x	x	x	x	x	
MG_359			x	x	x	x	x	x	
MG_360				x					
MG_361									
MG_362									
MG_363							x		
MG_522							x		
MG_364									
MG_365							x	x	
MG_366		x							
MG_367			x	x	x				
MG_368	x		x		x		x	x	
MG_369	x		x		x	x	x	x	
MG_370	x	x	x	x	x	x	x	x	x
MG_371									
MG_372	x	x	x		x		x	x	
MG_373									
MG_374									
MG_375							x		
MG_376	x		x		x	x	x	x	
MG_377									
MG_378									
MG_379			x		x		x		
MG_380		x	x	x	x	x	x	x	
MG_381									
MG_382			x		x		x	x	
MG_383			x		x		x	x	
MG_384			x		x				
MG_524									
MG_385	x	x	x	x	x	x	x	x	x
MG_386	x		x		x		x	x	
MG_387			x		x		x		
MG_388									
MG_389									
MG_390	x	x	x	x	x	x	x	x	x

	Bethesda	Rockville	Fujisawa	Rockville 2	Nashville	Stanford	Guelph	Valencia	Agreed
MG_391			x		x		x		
MG_392			x						
MG_393			x				x		
MG_394		x	x		x				
MG_395		x							
MG_396	x		x		x		x		
MG_397				x					
MG_398			x	x	x	x	x		
MG_399			x		x	x			
MG_400			x		x	x			
MG_401			x		x	x			
MG_402			x		x	x	x		
MG_403			x		x	x	x		
MG_404			x		x	x	x		
MG_405			x		x	x	x		
MG_406									
MG_407					x				
MG_408		x	x	x	x	x	x	x	
MG_409	x		x		x	x	x	x	
MG_410		x	x	x	x	x	x	x	
MG_411		x	x	x	x	x	x	x	
MG_412	x	x	x	x	x	x	x	x	x
MG_414		x		x					
MG_525		x		x					
MG_417									
MG_418									
MG_419	x		x		x		x	x	
MG_421		x	x		x	x	x	x	
MG_422									
MG_423		x							
MG_424									
MG_425			x		x				
MG_426		x					x		
MG_427	x		x		x	x	x	x	
MG_428	x	x	x	x	x	x	x	x	x
MG_429	x				x				
MG_430					x		x		
MG_431					x				
MG_432									
MG_433									
MG_434			x		x		x	x	
MG_435			x						
MG_437				x	x		x		
MG_438	x	x	x	x	x	x	x	x	x
MG_439									

	Bethesda	Rockville	Fujisawa	Rockville 2	Nashville	Stanford	Guelph	Valencia	Agreed
MG_440		x							
MG_441									
MG_442	x	x	x		x		x	x	
MG_443									
MG_444									
MG_445			x					x	
MG_446									
MG_447	x		x		x	x	x	x	
MG_448			x		x	x	x	x	
MG_449		x		x					
MG_450									
MG_451									
MG_452		x		x					
MG_453			x		x		x	x	
MG_454	x		x	x	x	x	x	x	
MG_455									
MG_456				x					
MG_457			x		x	x			
MG_458			x		x		x		
MG_459									
MG_460	x			x	x	x	x	x	
MG_461									
MG_462									
MG_463			x	x	x	x			
MG_464	x		x		x		x		
MG_465			x				x		
MG_466							x		
MG_467	x	x	x		x	x	x	x	
MG_468	x	x	x		x	x	x	x	
MG_526	x		x		x	x	x	x	
MG_469			x		x			x	
MG_470	x	x	x		x	x	x	x	

Table 24. Comparing the gene deletions of the minimal gene sets. Light grey genes are unmodelled in the *M.genitalium* whole-cell model. Dark grey genes cause the simulation to crash when deleted on our implementation (using Matlab R2013B on University of Bristol BlueGem's supercomputer (Section 3.4)). x = a gene deletion required to produce the minimal gene set in the *M.genitalium* whole-cell model.

5.3.2 Analysing the Minimal Gene Sets

A comparison of six of the minimal gene sets (the protocell minimal gene sets (Nashville, Fujisawa) were excluded due to their much more reduced size) showed that they had 96 genes in common (Table 23, Common Column). Of these 96 genes, 87 were classified as essential by single gene deletion, eight were non-essential, and one gene was unmodelled (Table 6). The 87 essential genes identify the cellular functional groups that the six minimal gene sets agree are essential: DNA (repair, supercoiling, chromosome replication, synthesis and modification of nucleotides, sigma factors, ligation, transcription termination, and DNA polymerase); RNA (ribosome proteins, initiation factors for translation, tRNA modification, ribonucleases, and RNA polymerase); and cellular processes (protein folding and modification, shuttling of proteins, protein membrane transport, production and recycling of metabolic substrates and intermediates, redox signalling, oxidation stress response, and the pyruvate dehydrogenase complex). Of the eight non-essential genes, four (MG_048, MG_072, MG_170, MG_297) are associated with the SecYEG complex [171] (which moves protein across or inserts them into the cell membrane), while MG_172 removes the start codon that initiates protein synthesis from synthesised nascent proteins, MG_305 and MG_392 assist in late protein folding, and MG_425 processes ribosomal RNA precursors. Although these eight genes are classified as singly non-essential, by single gene deletion *in-silico* [6] and *in-vivo* [61], they all play a part in essential functions within the cell, hence their inclusion in published minimal gene sets.

A comparison of the genes deleted from the *M.genitalium* genome by all the eight minimal gene sets showed that they shared 14 gene deletions in common (Table 24, Agreed Column). The function of these genes includes: fructose import, activation of host immune response, chromosomal partition, amino acid transport, antibody binding, phosphonate transport, external DNA uptake, DNA repair, rRNA modification, membrane breakdown, toxin transport, quorum sensing, and a restriction enzyme. These have all been previously classified as non-essential for cell survival by single gene deletion *in-silico* [6] and *in-vivo* [61], hence their exclusion from the minimal gene sets. I deleted these 14 common genes to produce an ‘Agreed set’ which was simulated in addition to the minimal gene sets from the literature.

5.3.3 Testing the Minimal Gene Sets

I simulated each minimal gene set in the *M.genitalium* whole-cell model and found that every set, including the Agreed set, produced a non-dividing *in-silico* cell. Each minimal gene set was simulated ten times per experiment, and each experiment was repeated three times (Table 25). These cells showed faults in metabolism, resulting in no DNA replication, RNA production, protein production, growth, or successful cell division.

An initial analysis found that every one of the sets from the literature deleted essential genes (as previously defined in Section 1.3.2.a). The number of essential genes deleted varied depending on the set: Nashville deleted 121, Fujisawa deleted 112, Guelph deleted 107, Valencia deleted 69, Bethesda deleted 34, Rockville and Rockville 2 both deleted 9, and Stanford deleted 3. This follows a similar pattern to *in-silico* genome size (Section 5.3.1), protocell minimal gene sets removed the largest number of essential genes and comparative genomics removed greater numbers of essential genes the higher the number of genomes compared. However, single gene deletion minimal gene sets also removed essential genes. This is likely due to transposon mutagenesis issues at the time of the minimal gene set design. Rockville labelled six genes as non-essential and so were excluded from the minimal gene set in 1999, but were later found to be essential by Rockville 2 in 2006. Even as recently as 2016, Hutchison *et al.* [12] had to make major improvements to their transposon mutagenesis protocol while producing *JCVI-Syn3.0*, due to incorrect identification of essentiality. Stanford removed MG_203, MG_250, and MG_470, likely due to averaging multiple simulation's data together before computationally assessing (Section 4.4.1). These are three genes that my simulations found to be essential (Table 8).

Repetitions	1	2	3
Bethesda	No Division	No Division	No Division
Bethesda	No Division	No Division	No Division
Bethesda	No Division	No Division	No Division
Bethesda	No Division	No Division	No Division
Bethesda	No Division	No Division	No Division
Bethesda	No Division	No Division	No Division
Bethesda	No Division	No Division	No Division
Bethesda	No Division	No Division	No Division
Bethesda	No Division	No Division	No Division
Bethesda	No Division	No Division	No Division
Rockville	No Division	No Division	No Division
Rockville	No Division	No Division	No Division
Rockville	No Division	No Division	No Division
Rockville	-	No Division	No Division
Rockville	No Division	No Division	No Division
Rockville	-	No Division	No Division
Rockville	No Division	No Division	No Division
Rockville	No Division	No Division	No Division
Rockville	No Division	No Division	No Division
Rockville	No Division	No Division	No Division
Fujisawa	-	No Division	-
Fujisawa	-	-	-
Fujisawa	-	-	No Division
Fujisawa	No Division	-	-
Fujisawa	-	-	-
Fujisawa	No Division	No Division	-
Fujisawa	No Division	-	-
Fujisawa	-	-	-
Fujisawa	-	No Division	-
Fujisawa	-	-	-
Rockville 2	No Division	No Division	No Division
Rockville 2	No Division	No Division	No Division
Rockville 2	No Division	No Division	No Division
Rockville 2	No Division	No Division	No Division
Rockville 2	No Division	No Division	No Division
Rockville 2	No Division	No Division	No Division
Rockville 2	No Division	No Division	No Division
Rockville 2	No Division	No Division	No Division
Rockville 2	No Division	No Division	No Division
Rockville 2	No Division	No Division	No Division
Nashville	-	No Division	-
Nashville	-	No Division	-
Nashville	-	-	No Division
Nashville	-	-	-
Nashville	-	No Division	-

Repetitions	1	2	3
Nashville	-	-	-
Nashville	-	-	-
Nashville	-	-	-
Nashville	No Division	No Division	-
Nashville	-	-	-
Stanford	No Division	No Division	No Division
Stanford	No Division	No Division	No Division
Stanford	No Division	No Division	No Division
Stanford	No Division	No Division	No Division
Stanford	No Division	No Division	No Division
Stanford	No Division	No Division	No Division
Stanford	No Division	No Division	No Division
Stanford	No Division	No Division	No Division
Stanford	No Division	No Division	No Division
Stanford	No Division	No Division	No Division
Guelph	-	No Division	No Division
Guelph	No Division	No Division	No Division
Guelph	No Division	No Division	No Division
Guelph	No Division	No Division	No Division
Guelph	No Division	No Division	No Division
Guelph	No Division	No Division	No Division
Guelph	No Division	No Division	No Division
Guelph	No Division	No Division	No Division
Guelph	No Division	No Division	No Division
Guelph	No Division	No Division	No Division
Valencia	No Division	No Division	No Division
Valencia	-	No Division	No Division
Valencia	-	-	-
Valencia	-	-	No Division
Valencia	-	-	No Division
Valencia	-	-	-
Valencia	No Division	-	-
Valencia	-	No Division	-
Valencia	-	-	-
Valencia	-	-	-
Agreed	No Division	No Division	No Division
Agreed	No Division	No Division	No Division
Agreed	No Division	No Division	No Division
Agreed	No Division	No Division	No Division
Agreed	No Division	No Division	No Division
Agreed	No Division	No Division	No Division
Agreed	No Division	No Division	No Division
Agreed	No Division	No Division	No Division
Agreed	No Division	No Division	No Division
Agreed	No Division	No Division	No Division

Table 25. Simulating the minimal gene sets as represented in the literature in the *M.genitalium* whole-cell model. - = a failed simulation due to supercomputer error.

5.3.4 Repairing the Minimal Gene Sets

I reintroduced the essential genes to the minimal gene sets, in an attempt to restore *in-silico* division. Based on previous research (Section 4.4.7), I also reintroduced low essential genes (e.g. if the gene is dispensable in some contexts, i.e. redundant essential genes and gene complexes [10]), knowing that they would impact the *in-silico* cell's ability to divide. Each "repaired" minimal gene set was simulated ten times per experiment, and each experiment was repeated three times (Table 26). By reintroducing these genes, specific to each set (Table 27), each set produced a dividing cell *in-silico* (Table 28), including the Agreed set (Table 24).

Repetitions	1	2	3
Bethesda	Dividing	Dividing	Dividing
Bethesda	Dividing	Dividing	Dividing
Bethesda	Dividing	Dividing	Dividing
Bethesda	Dividing	Dividing	Dividing
Bethesda	Dividing	Dividing	Dividing
Bethesda	Dividing	Dividing	Dividing
Bethesda	Dividing	Dividing	Dividing
Bethesda	Dividing	Dividing	Dividing
Bethesda	Dividing	Dividing	Dividing
Bethesda	Dividing	Dividing	Dividing
Rockville	Dividing	Dividing	Dividing
Rockville	Dividing	Dividing	Dividing
Rockville	Dividing	Dividing	Dividing
Rockville	Dividing	Dividing	Dividing
Rockville	Dividing	Dividing	Dividing
Rockville	Dividing	Dividing	Dividing
Rockville	Dividing	Dividing	Dividing
Rockville	Dividing	Dividing	Dividing
Rockville	Dividing	Dividing	Dividing
Rockville	Dividing	Dividing	Dividing
Fujisawa	Dividing	Dividing	Dividing
Fujisawa	Dividing	Dividing	Dividing
Fujisawa	Dividing	Dividing	Dividing
Fujisawa	Dividing	Dividing	Dividing
Fujisawa	Dividing	Dividing	Dividing
Fujisawa	Dividing	Dividing	Dividing
Fujisawa	Dividing	No Division	Dividing
Fujisawa	Dividing	Dividing	Dividing
Fujisawa	Dividing	No Division	Dividing
Fujisawa	Dividing	Dividing	Dividing
Rockville 2	Dividing	Dividing	Dividing
Rockville 2	Dividing	Dividing	Dividing
Rockville 2	Dividing	Dividing	Dividing
Rockville 2	Dividing	Dividing	Dividing
Rockville 2	Dividing	Dividing	Dividing
Rockville 2	Dividing	Dividing	Dividing
Rockville 2	Dividing	Dividing	Dividing
Rockville 2	Dividing	Dividing	Dividing
Rockville 2	Dividing	Dividing	Dividing
Rockville 2	Dividing	Dividing	Dividing
Nashville	Dividing	Dividing	Dividing
Nashville	Dividing	Dividing	Dividing
Nashville	Dividing	Dividing	Dividing
Nashville	Dividing	Dividing	Dividing
Nashville	Dividing	Dividing	Dividing

Repetitions	1	2	3
Nashville	Dividing	Dividing	Dividing
Nashville	Dividing	Dividing	Dividing
Nashville	Dividing	No Division	Dividing
Nashville	Dividing	Dividing	Dividing
Nashville	No Division	Dividing	No Division
Stanford	Dividing	Dividing	Dividing
Stanford	Dividing	Dividing	Dividing
Stanford	Dividing	Dividing	Dividing
Stanford	Dividing	Dividing	Dividing
Stanford	Dividing	Dividing	Dividing
Stanford	Dividing	Dividing	Dividing
Stanford	Dividing	No Division	Dividing
Stanford	Dividing	Dividing	Dividing
Stanford	Dividing	Dividing	Dividing
Stanford	Dividing	Dividing	Dividing
Guelph	Dividing	Dividing	Dividing
Guelph	Dividing	Dividing	Dividing
Guelph	Dividing	Dividing	No Division
Guelph	Dividing	Dividing	Dividing
Guelph	Dividing	Dividing	Dividing
Guelph	Dividing	Dividing	Dividing
Guelph	Dividing	Dividing	Dividing
Guelph	Dividing	Dividing	No Division
Guelph	Dividing	Dividing	Dividing
Guelph	Dividing	Dividing	Dividing
Valencia	Dividing	Dividing	Dividing
Valencia	Dividing	Dividing	Dividing
Valencia	Dividing	Dividing	Dividing
Valencia	Dividing	Dividing	Dividing
Valencia	No Division	Dividing	Dividing
Valencia	Dividing	Dividing	Dividing
Valencia	Dividing	Dividing	Dividing
Valencia	Dividing	Dividing	Dividing
Valencia	Dividing	Dividing	Dividing
Valencia	Dividing	Dividing	Dividing
Agreed	Dividing	Dividing	Dividing
Agreed	Dividing	Dividing	Dividing
Agreed	Dividing	Dividing	Dividing
Agreed	Dividing	Dividing	Dividing
Agreed	Dividing	Dividing	Dividing
Agreed	Dividing	Dividing	Dividing
Agreed	Dividing	Dividing	Dividing
Agreed	Dividing	Dividing	Dividing
Agreed	Dividing	Dividing	Dividing

Table 26. Simulating the minimal gene sets with genes reintroduced to “repair” them in the *M.genitalium* whole-cell model. The reintroduced genes are listed in Table 27.

	Protocells		Comparative Genomics			Single Gene Knockouts		
	Nashville	Fujisawa	Guelph	Bethesda	Valencia	Stanford	Rockville 2	Rockville
MG_001	r	r						
MG_003	r	r						
MG_004	r	r						
MG_006	r	r	r					
MG_007	r	r	r	r				
MG_008	r	r						
MG_013	r	r	r		r			
MG_019	r	r						
MG_022	r	r	r	r	r			
MG_023	r		r					
MG_026		r	r					
MG_031	r	r	r	r				
MG_034	r	r	r	r	r			
MG_037	r	r	r	r	r			
MG_038	r		r		r			
MG_039	r	r	r	r	r	r	r	r
MG_041	r		r	r				
MG_042	r	r	r		r			
MG_043	r	r	r		r			
MG_044	r	r	r		r			
MG_045	r	r	r		r			
MG_047	r	r						
MG_049	r	r	r		r			r
MG_051	r	r	r	r	r		r	r
MG_053	r	r	r		r			
MG_058	r	r						
MG_066	r	r					r	
MG_069	r		r	r				
MG_071	r	r	r		r			
MG_077	r	r	r		r			
MG_078	r	r	r		r			
MG_079	r	r	r		r			
MG_080	r	r	r		r			
MG_084	r	r		r				
MG_090			r					
MG_091	r	r	r		r			
MG_093			r					
MG_094	r	r						
MG_098	r	r	r	r	r			

	Nashville	Fujisawa	Guelph	Bethesda	Valencia	Stanford	Rockville 2	Rockville
MG_099		r	r	r	r			
MG_100		r	r	r	r			
MG_102	r	r						
MG_104	r	r	r		r			
MG_106	r	r	r		r			
MG_107	r	r						
MG_111	r							
MG_112	r	r					r	
MG_114	r		r		r		r	
MG_118	r	r	r		r			
MG_124	r	r						
MG_128	r	r	r	r	r			
MG_137	r	r	r	r	r			
MG_141	r	r						
MG_145	r	r						
MG_153			r					
MG_159			r					
MG_164			r					
MG_169		r	r					
MG_171		r						
MG_174			r					
MG_177	r							
MG_179	r	r	r	r	r			
MG_180	r	r	r		r			
MG_181	r	r	r	r	r			
MG_182	r	r	r		r			r
MG_195		r	r					
MG_201	r	r	r					
MG_203	r	r	r		r	r		
MG_204	r	r	r		r			
MG_212	r	r	r					
MG_215	r		r					
MG_216	r		r					
MG_221	r	r	r		r			
MG_224	r	r						
MG_228	r	r	r					
MG_229	r	r	r					
MG_230	r	r	r	r				
MG_231	r	r						
MG_232			r					
MG_238	r	r	r		r		r	
MG_245	r	r	r		r			
MG_249	r							
MG_250	r	r				r		
MG_251			r					

	Nashville	Fujisawa	Guelph	Bethesda	Valencia	Stanford	Rockville 2	Rockville
MG_254	r	r						
MG_258		r						
MG_261	r	r						
MG_270	r	r	r		r			
MG_271	r	r	r		r		r	
MG_272	r	r	r		r			
MG_273	r	r	r		r			
MG_274	r	r	r		r			
MG_275	r	r	r		r			
MG_276	r	r	r		r			
MG_278	r	r	r		r			
MG_282	r	r						
MG_283			r					
MG_287	r	r	r		r			
MG_289	r	r	r		r	r	r	
MG_290	r	r	r		r	r	r	
MG_291	r	r	r		r	r	r	*r*
MG_295	r	r						r
MG_299	r	r	r		r			
MG_300	r							
MG_301	r							
MG_302	r	r	r	r	r			
MG_303	r	r	r	r	r			
MG_304	r	r	r	r	r			
MG_305	r	r						
MG_311				r				
MG_315	r	r	r	r		r	r	
MG_321	r	r	r	r	r			
MG_322	r	r	r		r			
MG_323	r	r	r	r	r			
MG_325			r					
MG_330	r	r	r		r			
MG_340	r							
MG_341	r							
MG_342	r	r	r	r	r			
MG_345								r
MG_347	r	r	r		r			
MG_351			r					
MG_357	r	r	r		r			
MG_363			r					
MG_365			r		r			
MG_367	r	r					r	
MG_368	r	r	r	r	r			
MG_372	r	r	r	r	r			r
MG_375			r					

	Nashville	Fujisawa	Guelph	Bethesda	Valencia	Stanford	Rockville 2	Rockville
MG_379	r	r	r					
MG_382	r	r	r		r			
MG_383	r	r	r		r			
MG_384	r	r						
MG_387	r	r	r					
MG_394	r	r						r
MG_396	r	r	r	r				
MG_407	r							
MG_419	r	r	r	r	r			
MG_424	r							
MG_426			r					r
MG_427	r	r	r		r	r		
MG_429	r			r				
MG_430	r		r					
MG_431	r							
MG_434	r	r	r		r			
MG_435		r						
MG_437	r		r				r	
MG_444					r			
MG_445		r						
MG_453	r	r	r		r			
MG_458	r	r	r					
MG_465		r	r					
MG_466			r					
MG_469	r	r			r			
MG_470	r	r	r	r	r	r		r
MG_473	r	r	r	r	r			
MG_481	r	r	r	r	r			
MG_517	r	r	r	r	r			
MG_522			r					

Table 27. Comparing the gene reintroductions made to make the minimal gene sets produce *in-silico* dividing cells. Light grey genes are low essential genes in the *M.genitalium* whole-cell model. r = a gene reintroduction.

	Design approach	*In-silico* gene deletions (cell did not divide)	*In-silico* gene deletions (cell divided)	Genes reintroduced	Size of *in-silico* genome
Nashville	Protocell	270	142	128	217
Fujisawa	Protocell	261	141	120	218
Guelph	Comparative Genomics	241	129	112	230
Valencia	Comparative Genomics	185	110	75	249
Stanford	Single Gene Deletions	117	109	8	250
Bethesda	Comparative Genomics	118	82	36	277
Rockville 2	Single Gene Deletions	57	45	12	314
Rockville	Single Gene Deletions	53	43	10	316
Agreed	-	14	13	1	346

Table 28. Minimal gene sets that produce dividing *in-silico* cells. After the reintroduction of essential and low essential genes.

To repair the Agreed set, one low essential gene had to be reintroduced (MG_291), which is involved in phosphonate transport with two other genes (MG_289, MG_290). The Agreed set also removed MG_412, a gene that along with MG_410 and MG_411, is involved with phosphate transport. By disrupting both these processes, the Agreed set produced an *in-silico* cell that did not have a source of phosphate. This relationship has been established previously (Section 4.4.7, Table 16). MG_291 had to be reintroduced in every minimal gene set apart from Bethesda, which does not remove phosphate transport (MG_410, MG_411, MG_412). MG_289 and MG_290 also had to be reintroduced in six of the minimal gene sets.

I identified 31 genes that were reintroduced into at least five of the minimal gene sets (Table 29). 26 were classified as essential and 5 as low essential. The cellular functional groups that the minimal gene sets needed reintroducing were: DNA (polymerase delta / gamma / tau subunits, introduction of thymidine into DNA, rescue of pyrimidine bases for nucleotide synthesis, chromosome segregation); RNA (polymerase subunit delta, tRNA modification, 50S and 30S ribosomal subunits); transporters (cobalt, phosphonate, potassium); production (NAD, flavin, NADP, fatty acid/phospholipids); and dehydrogenation (glycerol and alpha-keto acids).

1	MG_022	11	MG_203	21	MG_321
2	MG_034	12	MG_238	22	MG_323
3	MG_037	13	MG_271	23	MG_342
4	MG_039	14	MG_289	24	MG_368
5	MG_051	15	MG_290	25	MG_372
6	MG_098	16	MG_291	26	MG_419
7	MG_128	17	MG_302	27	MG_427
8	MG_137	18	MG_303	28	MG_470
9	MG_179	19	MG_304	29	MG_473
10	MG_181	20	MG_315	30	MG_481
				31	MG_517

Table 29. 31 genes that were reintroduced into at least five minimal gene sets. They are all classified as essential (previously classified by single gene deletion *in-silico* [8,31]), apart from MG_039, MG_289, MG_290, MG_291, MG_427, which are classified as low essential (previously classified by single gene deletion *in-silico* [31]).

Of the 26 essential genes, 19 did not need to be reintroduced into the single gene deletion minimal gene sets (Stanford, Rockville, Rockville 2) as they had not been previously deleted. Two of these 19 genes, MG_137 (involved in cell wall production of mycobacteria) and MG_517 (produces fundamental components for plasma membrane stability) are specifically essential for *Mycoplasma* species, which were not identified as essential by the other, non species-specific, methodologies. Additionally, five of the 19 genes were involved in cobalt transport. Cobalt is used in the cell to produce cobalamin (otherwise known as vitamin B12), an enzyme cofactor which increases the reaction rates of DNA synthesis, fatty acid metabolism, and amino acid metabolism. This was also not identified as essential by the other design methodologies.

Interestingly, of the five low essential genes, Rockville and Bethesda did not delete four out of the five genes and Rockville 2 did not delete two of the five genes. It is likely that Bethesda did not remove the low essential genes due to the direct comparison with a closely related species, the genes with low essentiality being conserved to a greater degree than non-essential genes. I would speculate that Rockville removed less low essential genes than Rockville 2 due to Rockville using a transposon mutagenesis protocol with less precise results (as stated by Rockville 2 [83]), cells were grown in mixed

pools with DNA isolated from these mixtures rather than from isolated pure colonies of cells. This may have led to low essential genes that should have been labelled singly non-essential genes being incorrectly labelled as singly essential genes.

I investigated further the gene reintroductions to the protocell sets, as they could outline additional cellular requirements for the successful future unification of independent protocell systems. The genes reintroduced to the Nashville set repaired functions that had been reduced (translation, glycolytic process, protein folding) and restored functions that had been removed including: cell (division, cycle, transport, redox homeostasis), DNA (topological change, transcription), rRNA processing (including pseudouridine synthesis), protein transport, and cellular processes (carbohydrate metabolic, glycerol metabolic, fatty acid biosynthesis, UMP salvage) (Supplementary Data 27). The glycolytic process had the most change, with 10 of 11 genes being reintroduced, and DNA repair had the least, with only one gene being reintroduced (MG_254), however, this did reintroduce single strand DNA break repair. The genes additionally reintroduced to the Fujisawa set included tRNA processing and protein folding (Supplementary Data 28), with 8 out of 10 DNA replication genes reintroduced.

The protocell minimal gene sets (Nashville and Fujisawa) produced the smallest genomes *in-silico* (Table 28, Figure 16), once they were made functional, but they required the most number of genes to be reintroduced. The comparative genomics minimal gene sets (Guelph, Valencia, Bethesda) produce larger genomes *in-silico* but required less genes to be reintroduced. The smaller the number of genomes compared in their design, the less genes had to be reintroduced to make dividing *in-silico* cells. Interestingly, Stanford produced a smaller *in-silico* genome than Bethesda, as the Stanford minimal gene set did not target unmodelled gene deletions and only required eight genes to be reintroduced.

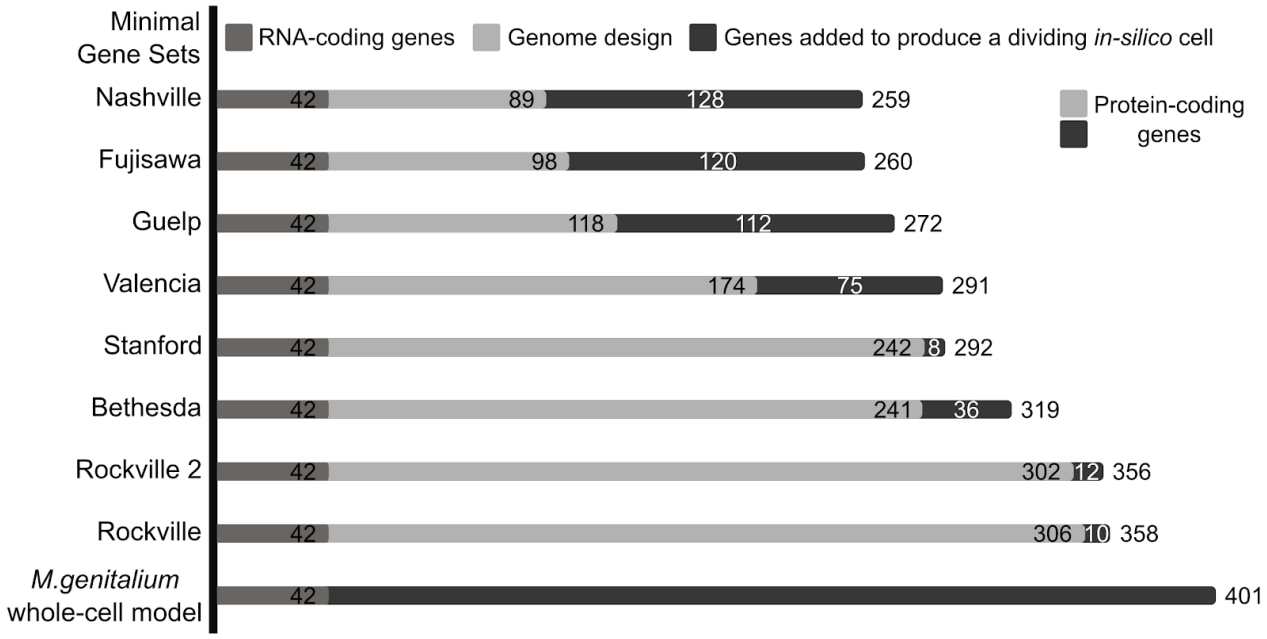

Figure 16. Repairing minimal gene sets from the literature to produce dividing *in-silico* cells. Minimal gene sets are shown from a range of design methodologies, produced across the past twenty years, and organised by the size of genome they produce in the whole-cell model. None of the original genome designs (light grey bars) produced a dividing *in-silico* cell in the *M.genitalium* whole-cell model. By reintroducing essential and low essential genes (black bars) I produced dividing *in-silico* cells.

5.4 Discussion

I tested minimal gene sets from the literature [6,55,59–61,74–76] for the first time, finding that they produced *in-silico* cells that did not divide, but by reintroducing specific essential and low essential genes the sets could produce dividing *in-silico* cells.

It currently appears that there is little to be learned for minimal genome research from modelling protocell minimal gene sets in the *M.genitalium* whole-cell model, given the amount of genetic modifications required to produce a viable *in-silico* cell. The comparative genomics minimal gene sets are closest to producing dividing cells *in-silico* when a lower number of genomes are compared. The single gene deletion minimal gene sets required the least genes to be reintroduced, so could be considered closest to viable *in-silico* minimal genomes from the outset, but without the ability to identify low essential genes, minimal gene sets designed with this methodology will still require correcting.

The identification of genes to reintroduce was straightforward given the previous labelling of *M.genitalium in-silico* essential and low essential genes, which would have been required if the information had not already been available. The results from this work reinforce a broader trend of moving away from universal minimal genomes [81] to species-specific minimal gene sets and minimal genomes (Sections 1.3.2.c, 1.3.5), and the need to specifically identify low essential genes and their interactions.

This research has the same limitations associated with the use of the *M.genitalium* whole-cell model, and the same uncertainty around the impact of the unmodelled genes on *in-vivo* experiments, as stated previously (Section 4.5).

This research advances minimal genome design. The computational predictions I have produced need to be tested in living cells, but with the advancement of gene synthesis and genome transplantation in other *Mycoplasma* species [11,62,113,114,172,173] this is becoming a more realistic proposition for *Mycoplasma* researchers.

Chapter 6 - *E.coli* Genome Engineering: *in-silico* and *in-vivo*

6.1 Aims

The *E.coli* whole-cell model being developed by the Covert Lab is the second whole-cell model [152] and presents the first opportunity to test whole-cell model results *in-vivo*, due to the difficulty of using *M.genitalium in-vivo* (Section 1.3.2.c). I installed the *E.coli* whole-cell model on the University of Bristol's BlueCrystal supercomputer, in the first steps towards combining the Minesweeper algorithm with the *E.coli* whole-cell model.

6.2 Implementing and Running the *E.coli* whole-cell Model

6.2.1 Implementing the *E.coli* whole-cell Model

The *E.coli* whole-cell model (Section 1.3.4.a) is currently under active development at the Covert Group, Stanford. They have made a release snapshot of it available for researchers outside of the group in advance of its publication. The *M.genitalium* whole-cell model was difficult to use even in its published state. Although the *E.coli* whole-cell model shows remarked improvement in usability, it is still at an early stage of development and implementing it on the University's BlueCrystal supercomputer took several months.

6.2.2 Issues with Installing the *E.coli* whole-cell Model

The current installation instructions (Oct 29th 2019) (github.com/CovertLab/WholeCellEcoliRelease/blob/master/docs/create-pyenv.md) for the *E.coli* whole-cell model have improved from the original drafts (June 12th 2019), but these instructions are still based on a Slurm rather than PBS supercomputer queueing system and assume advanced user controls which are not available to users of the University of Bristol's BlueCrystal.

The *E.coli* whole-cell model uses FireWorks (Section 6.2.3), a workflow management tool, to organise and conduct simulations. This requires the creation of an online database (specifically a MongoDB database) where information can be stored and retrieved by different computers running FireWorks. The provider of online databases

(MLab) used by the Covert Lab has been acquired since they started development. The implementation of the newly renamed product MongoDB Atlas is substantially different from MLab's previous offering and led to a host of issues. The solution I found was to use Heroku (a cloud application platform) to create my own version of MLab's online database (elements.heroku.com/addons/mongolab) and then proceed with the installation as described by the Covert Group.

To run FireWorks, you need to create .yaml files (detailed in Section 6.2.3) that define where the online database is and what information is required by the supercomputer to run simulations. Compared to the original documentation, I had to create a PBS queue version of the my_qadapter.yaml file. In addition, I had to add additional arguments to scripts that start the FireWorks process, giving the location of the .yaml files:

```
LAUNCHPAD_FILE='/newhome/jr0904/wholecell/WholeCellEcoliRelease/wholecell/fireworks/my_launchpad.yaml'

qlaunch -r -l my_launchpad.yaml -w my_fworker.yaml -q my_qadapter.yaml ..
```

There was also a step missing from the documentation due to structural differences between the development version of the model and the release snapshot. The release snapshot version of the model could not find the files it needed to run simulations, so I had to tell the model where its main directories were, using one of the installed python tools:

```
pyenv virtualenvwrapper
add2virtualenv .
```

Likewise, the model could not find the tools it needed from the supercomputer to run simulations. I could not tell the model where to find the files directly, as that would require changing a number of different parts of the model to search in specific BlueCrystal locations, rather than conduct a general search around the supercomputer. Thanks are due to Dee (Ms Dianaimh Greene) of the Advanced Computing Research Centre, Bristol, who confirmed my hypothesis and provided the syntax for solving this

final issue. This resulted in step four of the installation process, modifying the startup process to load specific tools from the supercomputer. As these tools (git and gcc) are pre-loaded, the model is capable of finding them through general searching:

```
~/.bash_profile:
module add tools/git-2.22.0
module add languages/gcc-7.1.0
```

6.2.3 Using the FireWorks workflow tool

FireWorks is an open-source tool for defining, managing, and executing workflows, compatible with Python and supercomputer queueing systems [174,175]. Workflows are defined as a number of tasks (Firetasks) within a hierarchy of jobs (Fireworks). These are uploaded to, stored, and managed on an online database (the LaunchPad). Computers or supercomputers (Fireworkers) connect to the online database (the Launchpad), retrieve jobs (Fireworks), execute them either locally or on the supercomputer queuing system, and record the results on the online database. The Fireworkers can be set to keep fetching and launching Fireworks until the workflow is complete. This allows you to have multiple supercomputers working together to complete the same workflow, running Fireworks in parallel. Workflows can also be paused and restarted, and Fireworks rerun, without causing errors.

To run FireWorks, you have to create three .yaml files. The first, my_fworker.yaml, names and allocates the current computer as a Fireworker:

```
name: default fireworker
category: ''
query: '{}'
```

The second, my_launchpad.yaml, provides the directions and login information for the online database (the Launchpad):

```
logdir: /newhome/jr0904/wholecell2/wcEcoli/wholecell/fireworks/logs/launchpad
host: ds345028.mlab.com
name: heroku_llcd42gj
username: squishybinary
password: *********
port: 45028
strm_lvl: INFO
user_indices: []
```

```
wf_user_indices: []
```

The third, my_qadapter.yaml, provides information about what queueing system the supercomputer uses, where the other .yaml files are located, and what information the supercomputer requires to accept a job:

```
logdir: /newhome/jr0904/wholecell2/wcEcoli/wholecell/fireworks/logs/qadapter
_fw_name: CommonAdapter
_fw_q_type: PBS
rocket_launch: rlaunch -w
/newhome/jr0904/wholecell2/wcEcoli/wholecell/fireworks/my_fworker.yaml -l
/newhome/jr0904/wholecell2/wcEcoli/wholecell/fireworks/my_launchpad.yaml
singleshot
nnodes: 1
ppnode: 1
walltime: '10:00:00'
queue: 'veryshort'
account: 'jr0904'
job_name: 'jr0904ecoli'
email: 'joshua.rees@bristol.ac.uk'
notification_options: 'bea'
pre_rocket: null
post_rocket: null
```

The online database (the Launchpad) records the status of Fireworks and workflows as: fizzled (the Firework has failed, it was executed but threw an error during the process, it can be rerun); running (the Firework is currently running); and completed (the Firework has successfully finished running). This can be checked at any time, from within the fireworks folder in the *E.coli* whole-cell model:

```
# Get a Launchpad report as a text file
lpad report -c fws -i months -n 2 > NAME_report_fws.txt
lpad report -c wflows -i months -n 2 > NAME_report_wflows.txt
lpad report -c launches -i months -n 2 > NAME_report_launches.txt

# Get Launchpad to assess errors in Fireworks and save to a text file
lpad introspect > NAME_introspection.txt

# Check Fizzled Fireworks and rerun
lpad get_fws -s FIZZLED -d all > NAME_fizzled.txt
lpad rerun_fws -s FIZZLED
```

6.2.4 Running the *E.coli* whole-cell model using FireWorks

The *E.coli* whole-cell model contains a script that translates designed simulations into workflows of Fireworks, which are subsequently uploaded to the online database, and a Fireworks command that sets the supercomputer to fetching and launching Fireworks, until the simulations and subsequent analysis are complete.

To create wild type *E.coli* simulations, a description of the simulation has to be provided to the fw_queue.py script, which translates it into a workflow. This is adapted from the Covert Group's simulation scripts github.com/CovertLab/WholeCellEcoliRelease/blob/master/runscripts/paper/paper_runs.sh. For example, to create four wild type simulations running for a single generation:

```
DESC="WildType_Test_1gen4seed" \
VARIANT="wild type" FIRST_VARIANT_INDEX=0 LAST_VARIANT_INDEX=0 \
SINGLE_DAUGHTERS=1 N_GENS=1 N_INIT_SIMS=4 \
MASS_DISTRIBUTION=1 GROWTH_RATE_NOISE=1 D_PERIOD_DIVISION=1 \
LAUNCHPAD_FILE='/newhome/jr0904/wholecell2/wcEcoli/wholecell/fireworks/my_launchpad.yaml' \
python /newhome/jr0904/wholecell2/wcEcoli/runscripts/fw_queue.py
```

To create *E.coli* gene knockout simulations, a different simulation description is provided to the fw_queue.py script, which translates it into a workflow. For example, to create one simulation, knocking out gene *metB* (EG10582) for two generations:

```
DESC="KO Name EG10582 KO Index 592" \
VARIANT="geneKnockout" FIRST_VARIANT_INDEX=592
LAST_VARIANT_INDEX=592 \
SINGLE_DAUGHTERS=1 N_GENS=2 N_INIT_SIMS=1 \
MASS_DISTRIBUTION=1 GROWTH_RATE_NOISE=1 D_PERIOD_DIVISION=1 \
LAUNCHPAD_FILE='/newhome/jr0904/wholecell2/wcEcoli/wholecell/fireworks/my_launchpad.yaml' \
python /newhome/jr0904/wholecell2/wcEcoli/runscripts/fw_queue.py
```

The Fireworks command that sets the supercomputer (Fireworker) to fetching and launching Fireworks, is called queue launcher.

```
qlaunch -r -l my_launchpad.yaml -w my_fworker.yaml -q my_qadapter.yaml rapidfire
--nlaunches infinite --sleep 30 --maxjobs_queue 100
```

These settings allow it to keep a maximum of 100 Fireworks in the supercomputer queue at any one time, to pause for thirty seconds between each Firework submission to the queue (allowing the supercomputer to keep up), and to launch an infinite number of Fireworks collected from the Launchpad (i.e. until the simulations and subsequent analysis are complete).

The simulation description and fireworks command are run on the supercomputer by writing them into a bash script, which I call FireWorksBoxes (github.com/squishybinary/Ecoli_whole-cell_model_FireWorksBoxes). This script is uploaded to the supercomputer and left running until all the simulations are complete.

6.3 *In-silico* Results

Following the successful setup of the *E.coli* whole-cell model and FireWorks on BlueCrystal, I successfully conducted wild type simulations and single gene knockout simulations of all 1214 implemented genes. The wildtype simulations were conducted for five generations (the minimum number of generations that the Covert Group has been conducting is four generations), whereas the single gene knockout simulations were conducted for two generations as a trade-off between computational space restrictions and the need for multiple generations to interpret gene essentiality (Section 4.4.8).

Figure 17 shows all 28 wild type *E.coli in-silico* cells simulated to date. All these simulations were continued for five generations, successfully producing dividing cells at each generation.

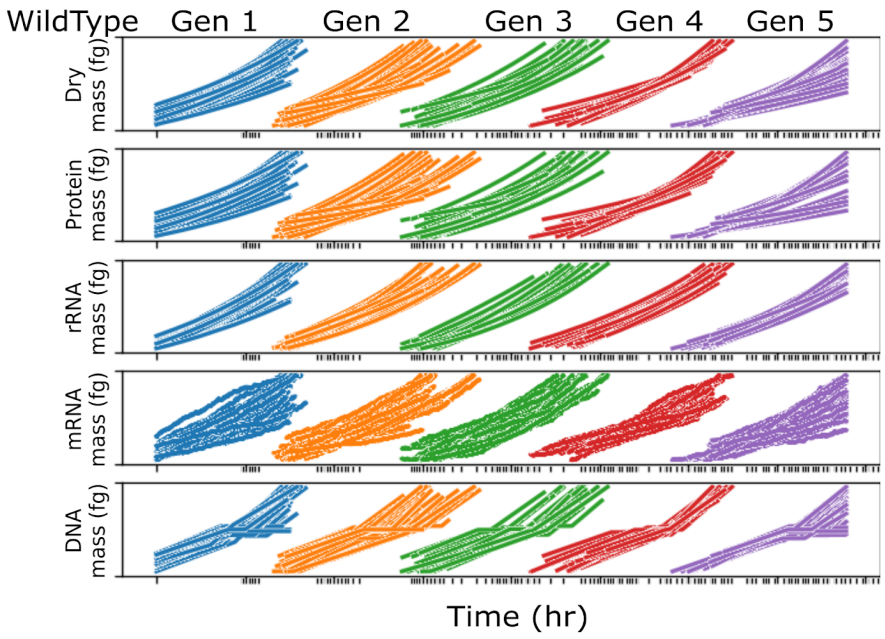

Figure 17. Overlay of 28 *E.coli* whole-cell model simulations. These are wild type simulations running for five generations. Changes in the mass of the cell, protein, rRNA, mRNA, and DNA are plotted.

Of the 1214 modelled genes that were knocked out in individual *in-silico* cells for two generations, 688 modelled genomes successfully produced output i.e. simulation data and automated graphs (Figure 18). These knockouts are non-essential genes, which do not prevent the cells from dividing regardless of the initial conditions of the cell. However, this indicates that the current release of the *E.coli* whole-cell model does not have a designated output for cells that fail to divide. Those that do not divide, do not produce a subsequent generation and so do not produce any automated graphs (which occurs after all generations are complete). Each of these 526 genes will have to be assessed individually to see whether the first or second generation failed to divide, and if a cause can be determined. Currently, I can infer that the genes are either high or complete essential genes that prevent division of the cell, prior or after a single generation of cells, but cannot identify the cause.

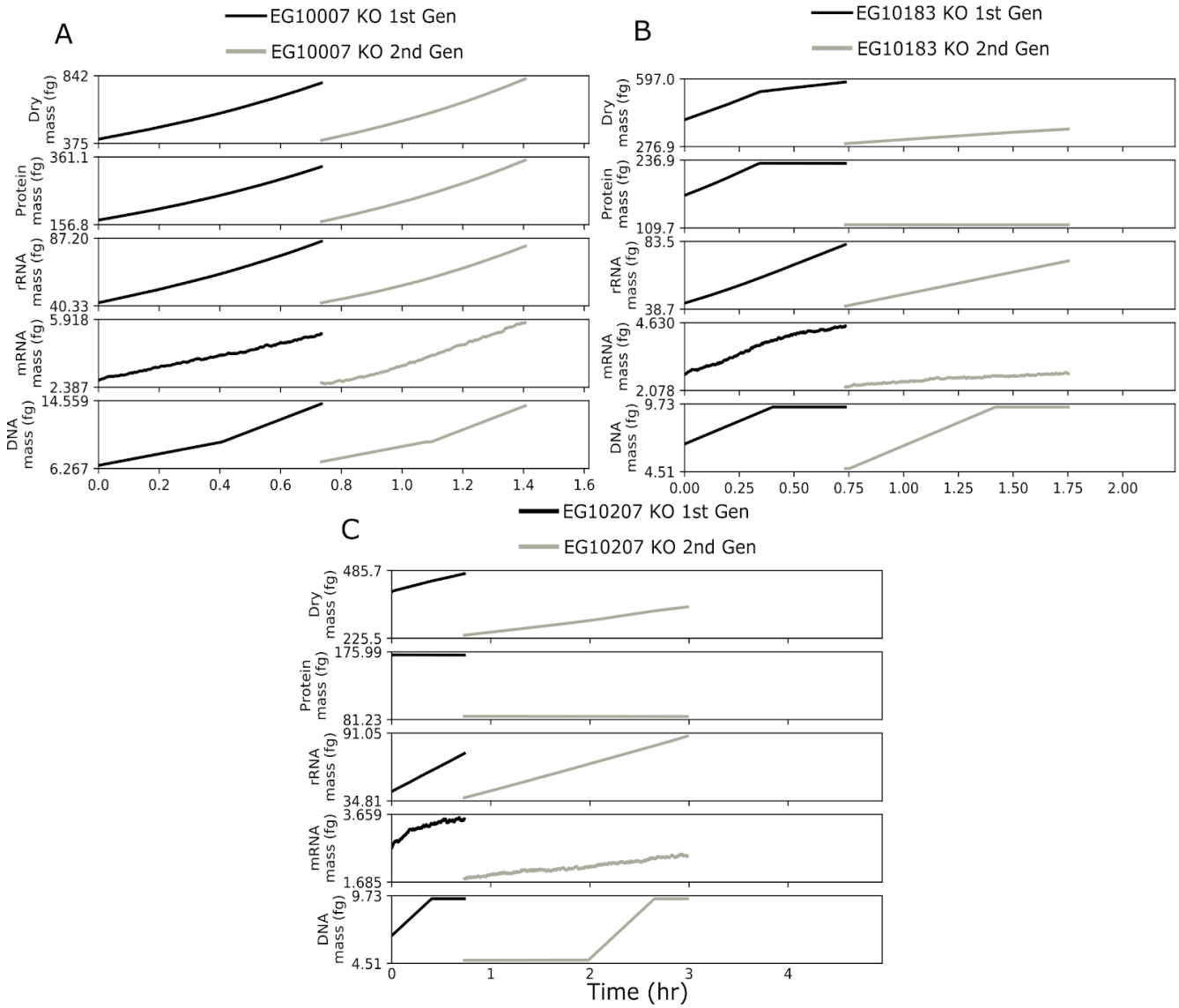

Figure 18. Comparing the plots of three single gene knockouts in the *E.coli* whole-cell model. These simulations ran for two generations and plotted changes in the mass of the cell, protein, rRNA, mRNA, and DNA with time.

Further simulations can be run in the future, to check how the single gene knockouts perform for one generation and five generations, to gain more understanding about each gene's essentiality. Currently this work has to wait, as I have encountered the more mundane issue of exceeding my supercomputer storage capacity.

For the 526 genes that did not produce any automated graphs one of four errors was recorded in the online database (the Launchpad):

- assert(np.all(moleculeCounts + np.dot(self.stoichMatrix, rxnFluxes) >= 0 (AssertionError)
- GLP_EFAIL: Solver failure
- Negative value(s): WATER in RnaDegradation, WATER in ProteinDegradation"
- UniqueObjectsContainerException: Object set is empty

These are all computational errors, rather than biological errors, which are difficult to interpret without the model creators' help. As previously mentioned, gene knockout has been implemented but not tested (Section 1.3.4.a). Following correspondence with the model creators, these sorts of computational bugs in knockout simulations are expected as they have not explored how the model behaves in this condition (*personal communication*).

6.3.1 *In-silico* Issues

Both Figure 17 and 18 show inconsistent axis plotting. Different labels are applied to the X axis (every hour, every half hour, or every three quarter hours) in different simulations. The Y axis labels only record the highest and lowest mass value across the five generations, the range of these Y axis labels is recorded in Table 30. This means the consistency seen in Figure 17 is not representative, as the plots are centred in a relative position to the highest and lowest mass values. This makes manual, visual interpretation difficult and automated comparison difficult to implement. These figures are produced automatically by the *E.coli* whole-cell model. Modifying this graphing will require familiarity with the model's data, akin to Section 3.3, which takes time to attain.

	Cell Mass Ranges	Protein Mass Ranges	rRNA Mass Ranges	mRNA Mass Ranges	DNA Mass Ranges
Lowest	241 - 360	98 - 158	25 - 39	1.4 - 2.4	4.2 - 6.2
Highest	755 - 922	315 - 378	81 - 100	4.8 - 7.7	13.2 - 16.4

Table 30. Range of Y axis values across 28 *E.coli* whole-cell model simulations. These are wild type simulations, running for five generations

For the three gene knockouts in Figure 18 (EG1007 (A), EG10183 (B), EG10207 (C)), rRNA mass is shown to increase in each generation of each knockout, but the time scales do not match, making actual interpretation difficult. Additionally, the B and C gene knockouts show similar protein mass behaviour in the second generation, but they actually have substantially different protein mass (109.7fg and 81.23fg), which is not easily observable from the plots. These two examples demonstrate that effective intercomparison of these single gene knockouts, as well as comparison to wild type cells' normal range of cellular performance, is not trivial using these automated graphs.

6.4 Future changes for using the *E.coli* whole-cell Model

The first change I would need to make is implementing graphing with absolute axis values. This requires identifying and extracting the simulation data that is currently used to generate the automated graphs. The absolute values for the axis can be initially decided by assessing the maximum values of several hundred wild type simulations (which can be adjusted as more simulations are conducted), and setting the minimum values at zero.

Secondly, I would need to modify the *E.coli* whole-cell model to allow non continuous multi-gene editing. Currently the model allows the deletion of single genes, and multiple genes within a range (i.e you can delete: gene 1, genes 2 - 5, genes 7-10; but you cannot delete genes 2, 4, and 5 or genes 7, 8, and 10). To do this, I need to create a version of geneknockout.py that allows the deletion of multiple genes that are not ordered with a range. This requires hijacking the FIRST_VARIANT_INDEX=N LAST_VARIANT_INDEX=N line passed to fwqueue.py, instead, using it to pass a row number. This subsequently requires writing a parsing function in geneknockout.py that intercepts the FIRST_VARIANT_INDEX=N LAST_VARIANT_INDEX=N row number, and

uses that to extract a gene deletion string from a gene deletions text file placed in a specific location, which is then passed to existing functions in geneknockout.py to process the gene deletions as normal. Alternatively, I could modify fwqueue.py to take lists instead of integers for the FIRST_VARIANT_INDEX=N LAST_VARIANT_INDEX=N, but this will likely have knock on effects elsewhere in the model as some functions rely on these values to be integers.

Thirdly, I would need to automate the creation of my version of the graphs and the creation of a summary text file. This summary text file needs to contain a description of the simulation (including the FireWorksBox file name), the genes deleted (and the associated row number), and whether the cell divided. This information is required so that Minesweeper can be run with the *E.coli* whole-cell model, and can assess the success of executed simulations and design the next cycle of simulations.

Finally, I would need to modify Minesweeper so that it can interpret this new format of summary text file, output FireWorksBox bash scripts (with the correct row and description) and the associated gene deletions text file, and be able to track the new results format in the log. These changes would allow me to run the Minesweeper minimal genome design algorithm with the *E.coli* whole-cell model.

6.5 Experimental Design for testing *E.coli* whole-cell model results *in-vivo*

6.5.1 Statement of Collaboration

I supervised two undergraduate students (Julia Needham and Cameron Matthews) for a summer studentship and am currently supervising a Masters by Research student (Jake Rightmyer). I wrote the *in-vivo* experimental design, shortlisted and selected the plasmid-based techniques, and trained and supervised the students. The students conducted all the laboratory work themselves. An overview of their work is included in this section.

6.5.2 Laboratory Work

One of the motivations for choosing *E.coli* as the next whole-cell model was the ability to test its *in-silico* predictions easily *in-vivo*. The difficulty of using *M.genitalium* in the lab [11,12] is one of the main issues with using the *M.genitalium* whole-cell model. I began making preparations in advance of receiving access to the *E.coli* whole-cell model, so that myself or other members of the group would be able to test the model's output in the lab. As the focus is currently on minimal genomes as a proof-of-concept, this would require conducting large-scale reductions of the *E.coli* genome. As highlighted previously (Section 1.3.2.d.1.a), there are a number of CRISPR-cas9 homologous recombination plasmid based techniques that show promise for doing this (Table 31), all of which benefit from reusable plasmids, elimination of selection steps, and absence of genomic scar sites compared to older techniques. CRISPR non homologous end joining techniques would save 124bp [112] per deletion, halving the cost of DNA per deletion, but they require an additional plasmid containing non homologous end joining *Mycoplasma* genes.

Authors	Jiang [108]	Li [107]	Pyne [105]	Reisch [106]	Zhao [109]	Zerbini [9]
Availability	AddGene	AddGene	AddGene	AddGene	Contact Lab	Contact Lab
BPs (>50% succ)	-	1000 - 12,000	8 - 818	500 - 3000	513 - 1400	30 - 2325
BPs (<1% succ)	-	-	19378	45000	-	-
Multiplexing	3 (47%)	2 (83%) 3 (23%)	-	3	-	2
Plasmids	2	2	3	2	1	2
Retargeting	Inverse PCR / BioBrick	Self Ligation GGA	pCRISPR Ligation / BsaI	CPEC / DpnI	Golden Gate Assembly	pCRISPR Ligation / BsaI
Plasmid Curing	37°C / pMB1	pGRB-p15 / Bla	37°C / None	37°C / pKDsgRNA-p15	37°C	-Cam / SacB(sucrose)
E.coli Strain	MG1655	EcPHE	DH5a	MG1655	MG1655 / DH5a	BL21(DE3)-ompA
Days (n per deletion)	2 + 2n	2 + 2n	-	3 + 2n	3n	2 + 2n
Antibiotics	Spectinomycin, Kanamycin	Chloramphenicol, Ampicillin	Ampicillin, Kanamycin, Chloramphenicol	Spectinomycin	Kanamycin	Kanamycin, Chloramphenicol
Methodology Criticism	Retargeting and curing are ambiguous	Chromosomal Lambda proteins required	No curing?	5 days to start, 3 days each for n deletion	Shorter homology arms, lower success rate (16% vs 100%)	Non-standard time calculation for methodology

Table 31. Comparison of CRISPR-cas9 homologous recombination techniques.

During my PhD, I supervised two undergraduate students who completed summer projects as part of the group. They attempted to test the protocol and results of Zhao *et*

al. [109], a CRISPR-cas9 - lambda protein single plasmid system combining CRISPR-cas9, lambda-red proteins, guideDNA and donorDNA. This edits bacterial genomes by using CRISPR-cas9 to make a double stranded break at a targeted location via the guideDNA (i.e. in this case the *poxB* gene), which is then repaired by homologous recombination using the donorDNA, promoted by lambda-red proteins. Successful gene deletions are then confirmed by colony PCR. Testing the protocol and replicating the results required growing up bacterial cultures, constructing the plasmid from four component parts (using Golden Gate Assembly in *E.coli* DH5a), and inserting the plasmid into *E.coli* MG1655 and triggering its activity.

A new Masters by Research student has recently joined the group. He is continuing to work on producing gene deletions in the *E.coli* genome, this time using the no-scar method [106] which is a two plasmid CRISPR-cas9 homologous recombination technique. One plasmid contains the CRISPR-cas9 and lambda-red proteins, which is inserted into the *E.coli* cell for the duration of the experimental series, and only denatured when all deletions have been completed. A second plasmid contains the guideDNA. A new version of this plasmid is constructed for each deletion and is inserted along with free floating donorDNA. This is expressed and then denatured. This technique was chosen over the previous system due to simpler plasmid construction, using a single antibiotic, and has plasmid curing steps for all plasmids involved (Table 31). This work is currently progressing and, global pandemic allowing, there is the potential for the Masters student to test *in-silico* predictions for multi-gene deletions *in-vivo* before the end of their project.

6.6 Discussion

Planned research that could be completed in the future includes: modifications to the *E.coli* whole-cell model (Section 6.4) to make it amenable to the Minesweeper algorithm; developing and standardising an analysis process for *E.*coli whole-cell model data; the production of multi-generational single gene knockouts for the *E.coli* whole-cell model using the analysis process; the application of the Minesweeper minimal genome design algorithm to the *E.coli* whole-cell model; and the testing of *E.coli in-silico* minimal genome predictions *in-vivo*.

Chapter 7 - General Discussion

7.1 Summary

Prior to the start of this PhD, the development of whole-cell mathematical models [6] and CRISPR-cas9 gene editing techniques [7–9], had produced the possibility of *in-silico* design and *in-vivo* editing for entire genome engineering. However, algorithms for genome design had not been developed.

During this PhD, I: created an analysis process for the *M.genitalium* whole-cell model; created Minesweeper, a minimal genome design algorithm; produced Minesweeper_256, an *in-silico M.genitalium* minimal genome; tested and reintroduced genes to eight minimal gene sets from the literature to produce dividing *in-silico M.genitalium* cells; and tested 1418 single gene knockouts, for two generations of cells, in the *E.coli* whole-cell model.

I made less progress towards a computational minimal genome for *E.coli* than I hoped. We received access to the whole-cell model in March 2019 but I could not start working with it until July due to being in the process of completing and submitting a paper. The implementation on the University's supercomputer proved difficult, even with the invaluable help of the Advanced Computing Research Centre at Bristol, simulations were not successfully running until mid-December. It is currently still not at the stage where Minesweeper can be applied (Section 6.4).

More time is required to produce the computational predictions from which large-scale reductions of an *E.coli* genome could be generated. I identified the best available *in-vivo* techniques to use. One of these was tested by undergraduate students, and another is in the process of being implemented by a Masters by Research student. Confidence in these techniques being fit for purpose was increased by the production of *E.coli* Syn61 using this family of techniques [72].

7.2 Future Work

Minesweeper is adaptable to future whole-cell models, as the algorithm interacts with the models only via the input of gene deletion lists and analysing the output. With future, multi-generational, whole-cell models I will have greater confidence that the algorithm has produced *in-silico* genome designs that will be viable *in-vivo*. This includes the *E.coli* whole-cell model at the Covert Lab, Stanford [152] and the *Mycoplasma pneumoniae* whole-cell model at the Karr Lab, Mount Sinai, New York [156].

It is difficult to see beyond the near-future of the genome design field. Gene synthesis will continue to fall in price as the technology improves [1] and projects like GP Write progress. But unless it does so drastically, the costs for genome-scale synthesis will likely still be outside the capabilities of any but the most well funded institutes (JCVI, Autodesk, Microsoft, BBN technologies) and large, international collaborations (Synthetic Yeast 2.0). This means optimal chassis development and novel microbial constructs will remain theoretical for the foreseeable future, at least for the majority of researchers (Section 1.3.2.d.1).

In regards to gene editing, CRISPR-cas9's capabilities currently exceed requirements. More still needs to be known about the potential off-target edits in prokaryotes and the success rates for conducting larger (tens of thousands of base pair) deletions, but it is otherwise fit for entire genome engineering [72,93]. Both the Church Lab and the Chin Lab are capable of genome recoding entire genomes with the tools available.

Finally, gene sequencing, through MinilON sequencing (although more error prone than Illumina sequencing), can be conducted at a low enough cost that any genome you want to sequence you feasibly can.

More granularly, the current available whole-cell models are likely to be the only ones available for the next five years due to their rate of development [150]. The Karr Lab is currently developing data aggregation tools and a whole-cell modelling programming language [151], necessary steps to democratise and speed up the production of whole-cell models but until these tools are functional and available, new models are

unlikely to appear. The work of the Covert Lab in creating the second whole-cell model [152] in much the same way they did the first is therefore necessary to keep the whole-cell model research community alive, sustainable, and publishing. Without new models of one or more cells that can be easily manipulated in the laboratory, continued interest in the field, and the ability for the field to compete for funding, will wither. Concerted effort needs to be made to expand the whole-cell modelling community to include new research teams who use rather than develop the models, particularly geneticists and synthetic biologists.

Even so, there is a chance the field may stagnate after any immediate research gains from the *E.coli* whole-cell model. Researchers may be in a holding pattern outside of a few established groups (JCVI progressing the understanding of minimal genomes, the Church Lab and the Chin Lab progressing the development of genome recoded strains, whole-cell model development continuing at the Covert Lab and Karr Lab) with the need for greater funding to edit genomes *in-vivo*, the requirement of greater knowledge to aid genome design *in-silico*, and the lack of new tools being developed in the short-term.

To avoid this, a new branch of research needs to be found for whole-cell models. This could be something as simple as a new genome design problem. There is plenty of research interest in designing optimal processes for a cell, especially in *E.coli*, which is feasible in whole-cell models. However, any optimality predictions produced will be tied to an even greater degree to the accuracy of the model *in-vivo*, which is yet to be tested (Section 4.5).

Alternatively, this could be the application of whole-cell models to the unification of protocells (Section 5.3.4). Analysis in Chapter 5 underlined the cellular components that protocell minimal gene sets had omitted, which were required to produce a dividing *in-silico* cell. In the future, when individual protocell systems are being introduced together in a cell-like environment, these omitted functions should be considered for introduction to produce desired, integrated behaviour. An in-development whole-cell model for an “archetypal bacterium” [151], that can represent a user-specified number of

genes, has the potential to be repurposed by the protocell community for modelling protocell systems; allowing for the continued search for omitted functions that can help glue individual systems together.

This new application for whole-cell models could also be found by using computational analysis to interrogate existing whole-cell model data (Section 4.4.2), finding new minima or key features that can advance genome design. New applications could also be found by developing and applying new algorithms to whole-cell models, including more advanced machine learning approaches. However, these algorithms will have to be developed by interdisciplinary teams, to be able to both understand how the algorithm works and how to interpret the data it produces. This is especially true if black box algorithms are used, as the output still needs to be connected to the input in an understandable, biological narrative to give confidence in the results. I would advise the preservation of real-world biological units (singular genes or entire genomes) to make this easier. Even basic tools, like principal component analysis* (PCA), that violate this rule make biological interpretation very difficult.

* PCA is a dimensionality reduction technique that attempts to represent features and patterns of large data sets in a new smaller package, creating new variables by combining and reducing the original data. It can be useful for finding patterns, but muddies interpretation to the point of making inference ill-advised to impossible (new data one is part gene x, part gene y, part relationship between genes a and c), and always requires secondary analysis. I dislike PCA. I have seen it used regularly, and incorrectly, for inference, when researchers should instead spend their time becoming intimately familiar with their datasets; gaining familiarity narrows the jump from pattern recognition to new understanding, and helps catch flaws and irregularities in datasets.

Successfully constructing a (likely minimal) genome *in-vivo*, from an *in-silico* design would change the methodology for future large-scale cellular research. It would move the field onto new genome design goals, increase momentum for using and improving the whole-cell model ecosystem, and could integrate with laboratory automation. Coupling established CRISPR-cas9 gene editing with biological programming

languages (such as Antha) and laboratory automation tools, directed by in-silico predictions, could revolutionise design-build-test cycles.

7.3 Conclusions

Whole-cell modelling had a vast amount of research potential but is not fully ready for use by the broader synthetic biology research community. The continued use of the *M.genitalium* whole-cell model is difficult to recommend, due to: it producing only single generations of cells, it not being solely based on *M.genitalium* data, and it being incredibly difficult to test genome engineering scale *in-silico* predictions in the laboratory. The *E.coli* whole-cell model should be adopted for research purposes but: is currently not published, only contains 1214 genes of ~4000, and is missing a number of sub-models. Additionally, analysis of the data produced by both models is difficult. This is due to the need to apply machine learning approaches to extract key information from the quantity of data produced, but machine learning approaches can be confounded by the complex nature of the data.

Single gene knockout studies will continue to incorrectly estimate minimal genome size, as low essential genes will be scored as non-essential [12,13,52] and if high essential genes are present they will be scored as essential. The presence of low essential genes in even the *M.genitalium in-silico* genome (Section 4.4.7, Table 18) provides further evidence that single gene knockout classifications are unreliable for genome minimisation, as they fail to take into account genomic context. It is likely low essential genes will be identified in the *E.coli in-silico* genome. These results reinforce a broader trend of moving away from universal minimal genomes [81] to species-specific research (Sections 1.3.2.c, 1.3.5), and the need to specifically identify, and avoid deleting, low essential genes and their interactions. Currently, our understanding of cellular life is not adequate to design novel or reduced minimal cells without further information.

Genome engineering using *in-silico* design closely combined with *in-vivo* editing would produce quicker, cheaper and more predictable laboratory results than currently possible, opening up genome design research to broader and more interdisciplinary research communities. However, the cost of each deletion is still high enough to make

the production of minimal genomes out of the scope for many individual PhD students and postdoctoral researchers.

As stated at the beginning of this thesis, tailoring entire genomes to produce custom-made cells is now on the horizon. If this is the case, *E.coli* is probably the bacterial species it will happen in first. Minesweeper in combination with the *E.coli* whole-cell model, with its testing over multiple generations and basis in species-specific data, should be able to produce *in-silico* minimal genomes that are viable *in-vivo*. The creation of *E.coli* Syn61 [72] has increased the research community's confidence in successfully conducting large-scale genetic edits using CRISPR-cas9 based methods. The process of *in-silico* design and *in-vivo* editing from metabolic engineering, given some more development time, will be available for genome engineering in *E.coli*. This would be *in-silico* design and *in-vivo* editing at a greater scale and depth (number and size of genetic edits) than ever seen before.

Bibliography

1. Carr, P. A. & Church, G. M. Genome engineering. *Nat. Biotechnol.* **27**, 1151–1162 (2009).
2. Kim, T. Y., Sohn, S. B., Kim, Y. B., Kim, W. J. & Lee, S. Y. Recent advances in reconstruction and applications of genome-scale metabolic models. *Curr. Opin. Biotechnol.* **23**, 617–623 (2012).
3. Xu, N., Ye, C. & Liu, L. Genome-scale biological models for industrial microbial systems. *Appl. Microbiol. Biotechnol.* **102**, 3439–3451 (2018).
4. Nielsen, J. & Keasling, J. D. Engineering Cellular Metabolism. *Cell* **164**, 1185–1197 (2016).
5. Waltemath, D. & Wolkenhauer, O. How Modeling Standards, Software, and Initiatives Support Reproducibility in Systems Biology and Systems Medicine. *IEEE Trans. Biomed. Eng.* **63**, 1999–2006 (2016).
6. Karr, J. R. *et al.* A whole-cell computational model predicts phenotype from genotype. *Cell* **150**, 389–401 (2012).
7. Jiang, W. Y., Bikard, D., Cox, D., Zhang, F. & Marraffini, L. A. RNA-guided editing of bacterial genomes using CRISPR-Cas systems. *Nat. Biotechnol.* **31**, 233–239 (2013).
8. Zhao, D. *et al.* CRISPR/Cas9-assisted gRNA-free one-step genome editing with no sequence limitations and improved targeting efficiency. *Sci. Rep.* **7**, 16624 (2017).
9. Zerbini, F. *et al.* Large scale validation of an efficient CRISPR/Cas-based multi gene editing protocol in Escherichia coli. *Microb. Cell Fact.* **16**, 68 (2017).
10. Rancati, G., Moffat, J., Typas, A. & Pavelka, N. Emerging and evolving concepts in gene essentiality. *Nat. Rev. Genet.* **19**, 34–49 (2018).

11. Benders, G. A. *et al.* Cloning whole bacterial genomes in yeast. *Nucleic Acids Res.* **38**, 2558–2569 (2010).
12. Hutchison, C. A. *et al.* Design and synthesis of a minimal bacterial genome. *Science* **351**, 1414–U73 (2016).
13. Glass, J. I., Merryman, C., Wise, K. S., Hutchison, C. A., 3rd & Smith, H. O. Minimal Cells-Real and Imagined. *Cold Spring Harb. Perspect. Biol.* **9**, 1-12 (2017)
14. Serrano, L. Synthetic biology: promises and challenges. *Mol. Syst. Biol.* **3**, 158 (2007).
15. Huang, W. E. & Nikel, P. I. The Synthetic Microbiology Caucus: from abstract ideas to turning microbes into cellular machines and back. *Microb. Biotechnol.* **12**, 5–7 (2019).
16. O'Malley, M. A., Powell, A., Davies, J. F. & Calvert, J. Knowledge-making distinctions in synthetic biology. *Bioessays* **30**, 57–65 (2008).
17. Brophy, J. A. N. & Voigt, C. A. Principles of genetic circuit design. *Nat. Methods* **11**, 508–520 (2014).
18. Dzieciol, A. J. & Mann, S. Designs for life: protocell models in the laboratory. *Chem. Soc. Rev.* **41**, 79–85 (2012).
19. Calero, P. & Nikel, P. I. Chasing bacterial chassis for metabolic engineering: a perspective review from classical to non-traditional microorganisms. *Microb. Biotechnol.* **12**, 98–124 (2019).
20. Endy, D. Foundations for engineering biology. *Nature* **438**, 449-453 (2005).
21. Isaacs, F. J. *et al.* Precise manipulation of chromosomes in vivo enables genome-wide codon replacement. *Science* **333**, 348–353 (2011).
22. Hirokawa, Y. *et al.* Genetic manipulations restored the growth fitness of

reduced-genome Escherichia coli. *J. Biosci. Bioeng.* **116**, 52–58 (2013).

23. Kohman, R. E., Kunjapur, A. M., Hysolli, E., Wang, Y. & Church, G. M. From Designing the Molecules of Life to Designing Life: Future Applications Derived from Advances in DNA Technologies. *Angew. Chem. Int. Ed Engl.* **57**, 4313–4328 (2018).
24. Lajoie, M. J. *et al.* Genomically recoded organisms expand biological functions. *Science* **342**, 357–360 (2013).
25. Adams, B. L. The Next Generation of Synthetic Biology Chassis: Moving Synthetic Biology from the Laboratory to the Field. *ACS Synth. Biol.* **5**, 1328–1330 (2016).
26. Vickers, C. E., Blank, L. M. & Krömer, J. O. Grand challenge commentary: Chassis cells for industrial biochemical production. *Nat. Chem. Biol.* **6**, 875–877 (2010).
27. Foley, P. L. & Shuler, M. L. Considerations for the design and construction of a synthetic platform cell for biotechnological applications. *Biotechnol. Bioeng.* **105**, 26–36 (2010).
28. Xavier, J. C., Patil, K. R. & Rocha, I. Systems Biology Perspectives on Minimal and Simpler Cells. *Microbiol. Mol. Biol. Rev.* **78**, 487–509 (2014).
29. Thompson, D. B. *et al.* The Future of Multiplexed Eukaryotic Genome Engineering. *ACS Chem. Biol.* **13**, 313–325 (2018).
30. Feist, A. M. & Palsson, B. Ø. The growing scope of applications of genome-scale metabolic reconstructions using Escherichia coli. *Nat. Biotechnol.* **26**, 659–667 (2008).
31. Thiele, I. & Palsson, B. Ø. A protocol for generating a high-quality genome-scale metabolic reconstruction. *Nat. Protoc.* **5**, 93–121 (2010).
32. Paddon, C. J. *et al.* High-level semi-synthetic production of the potent antimalarial

artemisinin. *Nature* **496**, 528–532 (2013).

33. Pontrelli, S. *et al.* Escherichia coli as a host for metabolic engineering. *Metab. Eng.* **50**, 16–46 (2018).
34. Gu, Y. *et al.* Advances and prospects of Bacillus subtilis cellular factories: From rational design to industrial applications. *Metab. Eng.* **50**, 109–121 (2018).
35. Deltcheva, E. *et al.* CRISPR RNA maturation by trans-encoded small RNA and host factor RNase III. *Nature* **471**, 602 (2011).
36. Jinek, M. *et al.* A programmable dual-RNA-guided DNA endonuclease in adaptive bacterial immunity. *Science* **337**, 816–821 (2012).
37. Biggs, B. W., De Paepe, B., Santos, C. N. S., De Mey, M. & Kumaran Ajikumar, P. Multivariate modular metabolic engineering for pathway and strain optimization. *Curr. Opin. Biotechnol.* **29**, 156–162 (2014).
38. Chen, X., Li, S. & Liu, L. Engineering redox balance through cofactor systems. *Trends Biotechnol.* **32**, 337–343 (2014).
39. Conrado, R. J. *et al.* DNA-guided assembly of biosynthetic pathways promotes improved catalytic efficiency. *Nucleic Acids Res.* **40**, 1879–1889 (2012).
40. Gu, Y. *et al.* Rewiring the Glucose Transportation and Central Metabolic Pathways for Overproduction of N-Acetylglucosamine in Bacillus subtilis. *Biotechnol. J.* **12**, 1700020 (2017).
41. Hemberger, S. *et al.* RibM from Streptomyces davawensis is a riboflavin/roseoflavin transporter and may be useful for the optimization of riboflavin production strains. *BMC Biotechnol.* **11**, 119 (2011).
42. Liu, Y. *et al.* A dynamic pathway analysis approach reveals a limiting futile cycle in N-acetylglucosamine overproducing Bacillus subtilis. *Nat. Commun.* **7**, 11933

(2016).

43. Portnoy, V. A., Bezdan, D. & Zengler, K. Adaptive laboratory evolution--harnessing the power of biology for metabolic engineering. *Curr. Opin. Biotechnol.* **22**, 590–594 (2011).
44. Wisselink, H. W., Toirkens, M. J., Wu, Q., Pronk, J. T. & van Maris, A. J. A. Novel evolutionary engineering approach for accelerated utilization of glucose, xylose, and arabinose mixtures by engineered Saccharomyces cerevisiae strains. *Appl. Environ. Microbiol.* **75**, 907–914 (2009).
45. Wright, J. *et al.* Batch and continuous culture-based selection strategies for acetic acid tolerance in xylose-fermenting Saccharomyces cerevisiae. *FEMS Yeast Res.* **11**, 299–306 (2011).
46. Singh, R., Kumar, M., Mittal, A. & Mehta, P. K. Microbial enzymes: industrial progress in 21st century. *3 Biotech* **6**, 174 (2016).
47. Szostak, J. W., Bartel, D. P. & Luisi, P. L. Synthesizing life. *Nature* **409**, 387–390 (2001).
48. D'Elia, M. A., Pereira, M. P. & Brown, E. D. Are essential genes really essential? *Trends Microbiol.* **17**, 433–438 (2009).
49. Qian, W., Ma, D., Xiao, C., Wang, Z. & Zhang, J. The genomic landscape and evolutionary resolution of antagonistic pleiotropy in yeast. *Cell Rep.* **2**, 1399–1410 (2012).
50. Commichau, F. M., Pietack, N. & Stülke, J. Essential genes in Bacillus subtilis: a re-evaluation after ten years. *Mol. Biosyst.* **9**, 1068–1075 (2013).
51. Ryan, C. J., Krogan, N. J., Cunningham, P. & Cagney, G. All or nothing: protein complexes flip essentiality between distantly related eukaryotes. *Genome Biol.*

Evol. **5**, 1049–1059 (2013).

52. Dobzhansky, T. Genetics of natural populations; recombination and variability in populations of Drosophila pseudoobscura. *Genetics* **31**, 269–290 (1946).
53. Smalley, D. J., Whiteley, M. & Conway, T. In search of the minimal Escherichia coli genome. *Trends Microbiol.* **11**, 6–8 (2003).
54. Motter, A. E., Gulbahce, N., Almaas, E. & Barabási, A.-L. Predicting synthetic rescues in metabolic networks. *Mol. Syst. Biol.* **4**, 168 (2008).
55. Gil, R. The Minimal Gene-Set Machinery. *Reviews in Cell Biology and Molecular Medicine* **2**, 1-32 (2014).
56. Rees-Garbutt, J. *et al.* Designing minimal genomes using whole-cell models. *Nat. Commun.* **11**, 836 (2020).
57. Gil, R., Silva, F. J., Pereto, J. & Moya, A. Determination of the core of a minimal bacterial gene set. *Microbiol. Mol. Biol. Rev.* **68**, 518–+ (2004).
58. Fraser, C. M. *et al.* The minimal gene complement of Mycoplasma genitalium. *Science* **270**, 397–403 (1995).
59. Mushegian, A. R. & Koonin, E. V. A minimal gene set for cellular life derived by comparison of complete bacterial genomes. *Proc. Natl. Acad. Sci. U. S. A.* **93**, 10268–10273 (1996).
60. Hutchison, C. A. *et al.* Global transposon mutagenesis and a minimal mycoplasma genome. *Science* **286**, 2165–2169 (1999).
61. Glass, J. I. *et al.* Essential genes of a minimal bacterium. *Proc. Natl. Acad. Sci. U. S. A.* **103**, 425–430 (2006).
62. Gibson, D. G. *et al.* Complete chemical synthesis, assembly, and cloning of a Mycoplasma genitalium genome. *Science* **319**, 1215–1220 (2008).

63. Antczak, M., Michaelis, M. & Wass, M. N. Environmental conditions shape the nature of a minimal bacterial genome. *Nat. Commun.* **10**, 3100 (2019).

64. Breuer, M. *et al.* Essential metabolism for a minimal cell. *Elife* **8**, 1-77 (2019).

65. Blattner, F. R. *et al.* The complete genome sequence of Escherichia coli K-12. *Science* **277**, 1453–1462 (1997).

66. Freiberg, C. *et al.* Identification of novel essential Escherichia coli genes conserved among pathogenic bacteria. *J. Mol. Microbiol. Biotechnol.* **3**, 483–489 (2001).

67. Kang, Y. S. *et al.* Systematic mutagenesis of the Escherichia coli genome. *J. Bacteriol.* **186**, 4921–4930 (2004).

68. Baba, T. *et al.* Construction of Escherichia coli K-12 in-frame, single gene knockout mutants: the Keio collection. *Mol. Syst. Biol.* **2**, 1-11 (2006).

69. Butland, G. *et al.* eSGA: E. coli synthetic genetic array analysis. *Nat. Methods* **5**, 789–795 (2008).

70. Gerdes, S. Y. *et al.* Experimental determination and system level analysis of essential genes in Escherichia coli MG1655. *J. Bacteriol.* **185**, 5673–5684 (2003).

71. Mizoguchi, H., Mori, H. & Fujio, T. Escherichia coli minimum genome factory. *Biotechnol. Appl. Biochem.* **46**, 157–167 (2007).

72. Fredens, J. *et al.* Total synthesis of Escherichia coli with a recoded genome. *Nature* **569**, 514–518 (2019).

73. Fleischmann, R. D. *et al.* Whole-genome random sequencing and assembly of Haemophilus influenzae Rd. *Science* **269**, 496–512 (1995).

74. Tomita, M. *et al.* E-CELL: software environment for whole-cell simulation. *Bioinformatics* **15**, 72–84 (1999).

75. Forster, A. C. & Church, G. M. Towards synthesis of a minimal cell. *Mol. Syst. Biol.*

2, 45 (2006).

76. Huang, C. H., Hsiang, T. & Trevors, J. T. Comparative bacterial genomics: defining the minimal core genome. *Antonie Van Leeuwenhoek International Journal of General and Molecular Microbiology* **103**, 385–398 (2013).
77. Shuler, M. L., Foley, P. & Atlas, J. Modeling a minimal cell. in *Microbial Systems Biology* (ed. Navid, A.) vol. 881 573–610 (Humana Press, 2012).
78. Mushegian, A. The minimal genome concept. *Curr. Opin. Genet. Dev.* **9**, 709–714 (1999).
79. Fraser, C. M., Eisen, J. A. & Salzberg, S. L. Microbial genome sequencing. *Nature* **406**, 799–803 (2000).
80. Koonin, E. V. Comparative genomics, minimal gene-sets and the last universal common ancestor. *Nat. Rev. Microbiol.* **1**, 127–136 (2003).
81. Lagesen, K., Ussery, D. W. & Wassenaar, T. M. Genome update: the 1000th genome - a cautionary tale. *Microbiology-Sgm* **156**, 603–608 (2010).
82. Acevedo-Rocha, C. G., Fang, G., Schmidt, M., Ussery, D. W. & Danchin, A. From essential to persistent genes: a functional approach to constructing synthetic life. *Trends Genet.* **29**, 273–279 (2013).
83. Glass, J. I. *et al.* Essential genes of a minimal bacterium. *Proc. Natl. Acad. Sci. U. S. A.* **103**, 425–430 (2006).
84. Juhas, M., Eberl, L. & Glass, J. I. Essence of life: essential genes of minimal genomes. *Trends Cell Biol.* **21**, 562–568 (2011).
85. Reich, K. A. The search for essential genes. *Res. Microbiol.* **151**, 319–324 (2000).
86. Reuß, D. R. *et al.* Large-scale reduction of the Bacillus subtilis genome: consequences for the transcriptional network, resource allocation, and

metabolism. *Genome Res.* **27**, 289–299 (2017).

87. Iwadate, Y., Honda, H., Sato, H., Hashimoto, M. & Kato, J.-I. Oxidative stress sensitivity of engineered Escherichia coli cells with a reduced genome. *FEMS Microbiol. Lett.* **322**, 25–33 (2011).
88. Mizoguchi, H., Sawano, Y., Kato, J. & Mori, H. Superpositioning of Deletions Promotes Growth of Escherichia coli with a Reduced Genome. *DNA Res.* **15**, 277–284 (2008).
89. Moya, A. *et al.* Toward minimal bacterial cells: evolution vs. design. *FEMS Microbiol. Rev.* **33**, 225–235 (2009).
90. Lee, J. H. *et al.* Metabolic engineering of a reduced-genome strain of Escherichia coli for L-threonine production. *Microb. Cell Fact.* **8**, 2 (2009).
91. Martínez-García, E. & de Lorenzo, V. The quest for the minimal bacterial genome. *Curr. Opin. Biotechnol.* **42**, 216–224 (2016).
92. Sleator, R. D. The story of Mycoplasma mycoides JCVI-syn1.0: the forty million dollar microbe. *Bioeng. Bugs* **1**, 229–230 (2010).
93. Zhou, J., Wu, R., Xue, X. & Qin, Z. CasHRA (Cas9-facilitated Homologous Recombination Assembly) method of constructing megabase-sized DNA. *Nucleic Acids Res.* **44**, e124 (2016).
94. Lartigue, C. *et al.* Genome transplantation in bacteria: changing one species to another. *Science* **317**, 632–638 (2007).
95. Lartigue, C. *et al.* Creating bacterial strains from genomes that have been cloned and engineered in yeast. *Science* **325**, 1693–1696 (2009).
96. Baby, V. *et al.* Cloning and Transplantation of the Mesoplasma florum Genome. *ACS Synth. Biol.* **7**, 209–217 (2018).

97. Yu, B. J. *et al.* Minimization of the Escherichia coli genome using a Tn5-targeted Cre/loxP excision system. *Nat. Biotechnol.* **20**, 1018–1023 (2002).

98. Hashimoto, M. *et al.* Cell size and nucleoid organization of engineered Escherichia coli cells with a reduced genome. *Mol. Microbiol.* **55**, 137–149 (2005).

99. Posfai, G. *et al.* Emergent properties of reduced-genome Escherichia coli. *Science* **312**, 1044–1046 (2006).

100. Park, M. K. *et al.* Enhancing recombinant protein production with an Escherichia coli host strain lacking insertion sequences. *Appl. Microbiol. Biotechnol.* **98**, 6701–6713 (2014).

101. Murphy, K. C. Use of bacteriophage lambda recombination functions to promote gene replacement in Escherichia coli. *J. Bacteriol.* **180**, 2063–2071 (1998).

102. Costantino, N. & Court, D. L. Enhanced levels of lambda red-mediated recombinants in mismatch repair mutants. *Proc. Natl. Acad. Sci. U. S. A.* **100**, 15748–15753 (2003).

103. Sharan, S. K., Thomason, L. C., Kuznetsov, S. G. & Court, D. L. Recombineering: a homologous recombination-based method of genetic engineering. *Nat. Protoc.* **4**, 206–223 (2009).

104. Sander, J. D. & Joung, J. K. CRISPR-Cas systems for editing, regulating and targeting genomes. *Nat. Biotechnol.* **32**, 347–355 (2014).

105. Pyne, M. E., Moo-Young, M., Chung, D. A. & Chou, C. P. Coupling the CRISPR/Cas9 System with Lambda Red Recombineering Enables Simplified Chromosomal Gene Replacement in Escherichia coli. *Appl. Environ. Microbiol.* **81**, 5103–5114 (2015).

106. Reisch, C. R. & Prather, K. L. J. The no-SCAR (Scarless Cas9 Assisted

Recombineering) system for genome editing in Escherichia coli. *Sci. Rep.* **5**, 15096 (2015).

107. Li, Y. *et al.* Metabolic engineering of Escherichia coli using CRISPR-cas9 meditated genome editing. *Metab. Eng.* **31**, 13–21 (2015).

108. Jiang, Y. *et al.* Multigene Editing in the Escherichia coli Genome via the CRISPR-cas9 System. *Appl. Environ. Microbiol.* **81**, 2506–2514 (2015).

109. Zhao, D. *et al.* Development of a fast and easy method for Escherichia coli genome editing with CRISPR/Cas9. *Microb. Cell Fact.* **15**, 205 (2016).

110. Standage-Beier, K., Zhang, Q. & Wang, X. Targeted large-scale deletion of bacterial genomes using CRISPR-nickases. *ACS Synth. Biol.* **4**, 1217–1225 (2015).

111. Su, T. *et al.* A CRISPR-cas9 Assisted Non-Homologous End-Joining Strategy for One-step Engineering of Bacterial Genome. *Sci. Rep.* **6**, 37895 (2016).

112. Zheng, X., Li, S.-Y., Zhao, G.-P. & Wang, J. An efficient system for deletion of large DNA fragments in Escherichia coli via introduction of both Cas9 and the non-homologous end joining system from Mycobacterium smegmatis. *Biochem. Biophys. Res. Commun.* **485**, 768–774 (2017).

113. Karas, B. J. *et al.* Direct transfer of whole genomes from bacteria to yeast. *Nat. Methods* **10**, 410–412 (2013).

114. Tsarmpopoulos, I. *et al.* In-Yeast Engineering of a Bacterial Genome Using CRISPR/Cas9. *ACS Synth. Biol.* **5**, 104–109 (2016).

115. Wang, L. *et al.* Synthetic Genomics: From DNA Synthesis to Genome Design. *Angew. Chem. Int. Ed Engl.* **57**, 1748–1756 (2018).

116. Ostrov, N. *et al.* Design, synthesis, and testing toward a 57-codon genome.

Science **353**, 819–822 (2016).

117. Wang, H. H. *et al.* Programming cells by multiplex genome engineering and accelerated evolution. *Nature* **460**, 894–898 (2009).

118. Ma, N. J., Moonan, D. W. & Isaacs, F. J. Precise manipulation of bacterial chromosomes by conjugative assembly genome engineering. *Nat. Protoc.* **9**, 2285–2300 (2014).

119. Wang, K. *et al.* Defining synonymous codon compression schemes by genome recoding. *Nature* **539**, 59–64 (2016).

120. Aguilar Suarez, R., Stülke, J. & van Dijl, J. M. Less is more: towards a genome-reduced Bacillus cell factory for 'difficult proteins'. *ACS Synth. Biol.* **8**, 99–108 (2019).

121. Ara, K. *et al.* Bacillus minimum genome factory: effective utilization of microbial genome information. *Biotechnol. Appl. Biochem.* **46**, 169–178 (2007).

122. Morimoto, T. *et al.* Enhanced recombinant protein productivity by genome reduction in Bacillus subtilis. *DNA Res.* **15**, 73–81 (2008).

123. Napolitano, M. G. *et al.* Emergent rules for codon choice elucidated by editing rare arginine codons in Escherichia coli. *Proc. Natl. Acad. Sci. U. S. A.* **113**, E5588–97 (2016).

124. Richardson, S. M. *et al.* Design of a synthetic yeast genome. *Science* **355**, 1040–1044 (2017).

125. Henry, C. S., Overbeek, R. & Stevens, R. L. Building the blueprint of life. *Biotechnol. J.* **5**, 695–704 (2010).

126. Borkowski, O. *et al.* Cell-free prediction of protein expression costs for growing cells. *Nat. Commun.* **9**, 1457 (2018).

127. Ceroni, F. & Ellis, T. The challenges facing synthetic biology in eukaryotes. *Nat. Rev. Mol. Cell Biol.* **19**, 481–482 (2018).

128. Mol, M., Kabra, R. & Singh, S. Genome modularity and synthetic biology: Engineering systems. *Prog. Biophys. Mol. Biol.* **132**, 43–51 (2018).

129. Park, S. Y., Yang, D., Ha, S. H. & Lee, S. Y. Metabolic Engineering of Microorganisms for the Production of Natural Compounds. *Adv. Biosys.* **2**, 1700190 (2018).

130. Orth, J. D., Thiele, I. & Palsson, B. O. What is flux balance analysis? *Nat. Biotechnol.* **28**, 245–248 (2010).

131. Systems Biology UCSD Database. Systems Biology UCSD http://systemsbiology.ucsd.edu/InSilicoOrganisms/OtherOrganisms (2018).

132. Kanehisa, M. & Goto, S. KEGG: kyoto encyclopedia of genes and genomes. *Nucleic Acids Res.* **28**, 27–30 (2000).

133. Devoid, S. *et al.* Automated genome annotation and metabolic model reconstruction in the SEED and Model SEED. *Methods Mol. Biol.* **985**, 17–45 (2013).

134. Karp, P. D. *et al.* Pathway Tools version 19.0 update: software for pathway/genome informatics and systems biology. *Brief. Bioinform.* **17**, 877–890 (2016).

135. Aite, M. *et al.* Traceability, reproducibility and wiki-exploration for 'à-la-carte' reconstructions of genome-scale metabolic models. *PLoS Comput. Biol.* **14**, e1006146 (2018).

136. Dias, O., Rocha, M., Ferreira, E. C. & Rocha, I. Merlin: Metabolic models reconstruction using genome-scale information. *IFAC Proceedings Volumes* **43**,

120–125 (2010).

137. Olivier, B. G. MetaDraft. https://systemsbioinformatics.github.io/cbmpy-metadraft/ (2018).

138. Wang, H. *et al.* RAVEN 2.0: A versatile toolbox for metabolic network reconstruction and a case study on Streptomyces coelicolor. *PLoS Comput. Biol.* **14**, e1006541 (2018).

139. Machado, D., Andrejev, S., Tramontano, M. & Patil, K. R. Fast automated reconstruction of genome-scale metabolic models for microbial species and communities. *Nucleic Acids Res.* **46**, 7542–7553 (2018).

140. Mendoza, S. N., Olivier, B. G., Molenaar, D. & Teusink, B. A systematic assessment of current genome-scale metabolic reconstruction tools. *Genome Biol*. **20**, 158 (2019) .

141. Edwards, J. S. & Palsson, B. O. The Escherichia coli MG1655 in silico metabolic genotype: Its definition, characteristics, and capabilities. *Proc. Natl. Acad. Sci. U. S. A.* **97**, 5528–5533 (2000).

142. Oh, Y.-K., Palsson, B. O., Park, S. M., Schilling, C. H. & Mahadevan, R. Genome-scale reconstruction of metabolic network in Bacillus subtilis based on high-throughput phenotyping and gene essentiality data. *J. Biol. Chem.* **282**, 28791–28799 (2007).

143. Förster, J., Famili, I., Palsson, B. Ø. & Nielsen, J. Large-scale evaluation of in silico gene deletions in Saccharomyces cerevisiae. *OMICS* **7**, 193–202 (2003).

144. Schilling, C. H. *et al.* Genome-scale metabolic model of Helicobacter pylori 26695. *J. Bacteriol.* **184**, 4582–4593 (2002).

145. Shinfuku, Y. *et al.* Development and experimental verification of a genome-scale

metabolic model for Corynebacterium glutamicum. *Microb. Cell Fact.* **8**, 43 (2009).

146. Lloyd, C. J. Expanding the Scope of Genome-scale Models of Metabolism and Gene Expression. (UC San Diego, 2019).

147. Sanghvi, J. C. *et al.* Accelerated discovery via a whole-cell model. *Nat. Methods* **10**, 1192–1195 (2013).

148. Purcell, O., Jain, B., Karr, J. R., Covert, M. W. & Lu, T. K. Towards a whole-cell modeling approach for synthetic biology. *Chaos* **23**, 025112 (2013).

149. Kazakiewicz, D., Karr, J. R., Langner, K. M. & Plewczynski, D. A combined systems and structural modeling approach repositions antibiotics for Mycoplasma genitalium. *Comput. Biol. Chem.* **59 Pt B**, 91–97 (2015).

150. Szigeti, B. *et al.* A blueprint for human whole-cell modeling. *Curr Opin Syst Biol* **7**, 8–15 (2018).

151. Karr, J. Tools for building, simulating, analyzing whole-cell models. *Whole-Cell Modeling* https://www.wholecell.org/tools/ (2019).

152. Covert, M. *E.coli* Whole Cell Model. *Github* https://github.com/CovertLab/WholeCellEcoliRelease (2019).

153. Wang, L. & Maranas, C. D. MinGenome: An In Silico Top-Down Approach for the Synthesis of Minimized Genomes. *ACS Synth. Biol.* **7**, 462–473 (2018).

154. Price, M. N. *et al.* Mutant phenotypes for thousands of bacterial genes of unknown function. *Nature* **557**, 503–509 (2018).

155. Gibson, D. G. Programming biological operating systems: genome design, assembly and activation. *Nat. Methods* **11**, 521–526 (2014).

156. Karr, J. Comprehensive whole-cell computational models of individual cells. *Whole-Cell Modeling* https://www.wholecell.org/models/ (2019).

157. Waltemath, D. *et al.* Toward Community Standards and Software for Whole-Cell Modeling. *IEEE Trans. Biomed. Eng.* **63**, 2007–2014 (2016).

158. Minhee, H. & Mcconville, E. Wand 0.5.8. http://docs.wand-py.org/en/0.5.8/ (2020).

159. ImageMagick Studio, L. L. C. ImageMagick. https://imagemagick.org/ (2008).

160. Apweiler, R. *et al.* UniProt: the Universal Protein knowledgebase. *Nucleic Acids Res.* **32**, D115–9 (2004).

161. Marucci, L., Grierson, C., Chalkley, O., Rees, J. & Landon, S. Data from whole-cell modelling. doi:10.5523/BRIS.1JJ0FSZZRX9QF2LDCZ654QP454 (2019).

162. Knuth, D. E. *The Art of Computer Programming: Sorting and searching*. (Addison-Wesley, 1998).

163. Karr, J. *M.genitalium* Whole Cell Model. *Github*. https://github.com/CovertLab/WholeCell (2012)

164. Chalkley, O., Purcell, O., Grierson, C. & Marucci, L. The genome design suite: enabling massive in-silico experiments to design genomes. *bioRxiv.* doi:10.1101/681270 (2019).

165. Lin, H.-N. & Hsu, W.-L. Kart: a divide-and-conquer algorithm for NGS read alignment. *Bioinformatics* **33**, 2281–2287 (2017).

166. Back, T. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. (Oxford University Press, 1996).

167. Burgard, A. P., Vaidyaraman, S. & Maranas, C. D. Minimal reaction sets for Escherichia coli metabolism under different growth requirements and uptake environments. *Biotechnol. Prog.* **17**, 791–797 (2001).

168. Huynen, M. Constructing a minimal genome. *Trends Genet.* **16**, 116 (2000).

169. Kamat, S. S., Williams, H. J. & Raushel, F. M. Intermediates in the transformation of phosphonates to phosphate by bacteria. *Nature* **480**, 570–573 (2011).

170. Grosjean, H. *et al.* Predicting the minimal translation apparatus: lessons from the reductive evolution of mollicutes. *PLoS Genet.* **10**, e1004363 (2014).

171. du Plessis, D. J. F., Nouwen, N. & Driessen, A. J. M. The Sec translocase. *Biochim. Biophys. Acta* **1808**, 851–865 (2011).

172. Gibson, D. G. *et al.* Creation of a Bacterial Cell Controlled by a Chemically Synthesized Genome. *Science* **329**, 52–56 (2010).

173. Gibson, D. G. *et al.* One-step assembly in yeast of 25 overlapping DNA fragments to form a complete synthetic Mycoplasma genitalium genome. *Proc. Natl. Acad. Sci. U. S. A.* **105**, 20404–20409 (2008).

174. Jain, A. *et al.* FireWorks: a dynamic workflow system designed for high-throughput applications. *Concurr. Comput.* **27**, 5037–5059 (2015).

175. Jain, A. *et al*. FireWorks 1.9.5. https://materialsproject.github.io/fireworks/ (2020).

176. Johnson, C. & Donner, R. *Minesweeper*. (1990).

Appendices

9.1 Formalising Minesweeper by Analogy

In an attempt to explain the stages of the algorithm, I likened it to the game of Minesweeper [176], envisioning removing genes from the *in-silico* cell to see if it causes cell death as equivalent to pressing tiles to potentially reveal a mine.
In this analogy, the first stage is easy. We press each tile one by one (one tile equals a single gene). If a tile is hiding a mine (i.e. gene removal kills the cell), we lock that tile and prevent it from being pressed again (the gene is no longer considered for deletion).

The second stage is trickier because we have found all the obvious mines. Some tile presses (gene deletions) only trigger mines in combination with another, or can be primed or deactivated by another tile being pressed first (an associated gene being deleted previously). To find these conditional mines, we have to press as many tiles as we can without setting them off, and note when we do. In the second step, we press the remaining unlocked tiles (singly non-essential genes) in big groups, the largest being 100% of the unlocked tiles, the smallest being 12.5%.

The third stage takes all the groups of tiles that did not contain a mine (prevent the *in-silico* cell from dividing) from the second stage, combines them in all possible non-overlapping combinations, and tests them all.

The fourth stage takes the biggest combination of tiles that did not set off a conditional mine (the largest group of gene deletions), and locks them in. The remaining unlocked tiles (singly non-essential genes that are yet to be deleted) are divided into eight groups. The eight groups are tested individually and in combination, from two to seven of the eight groups, in addition to the locked tiles (while also deleting the largest group of gene deletions), to find the groups that do contain conditional mines. This step is repeated as we find lock in more tiles and isolate the location of conditional mines, until we cannot press any more tiles without setting off a mine (prevent the *in-silico* cell from dividing).